Universidade do Minho

MAP-i Doctoral Program in Computer Science

A Virtual Factory for Smart City Service Integration

Author:
María Guillermina Cledou

Supervisor:
Luis Manuel Dias Coelho Soares Barbosa
Elsa Estevez

Version: 9th January 2019

Acknowledgements

I immensely thank my parents that support me unconditionally in every decision I take, and my grandma who called me every day since I moved away.

I would like to thank my supervisors, Elsa and Luis, for their support and guidance through out these years. To Elsa, who encourage me to pursue this PhD, and who I admire. Thank you for always motivating me to go beyond my comfort zone to become a better professional, for your trust, and for adopting me since day zero and treating me like a daughter when I was far away from home. To Luis, who guided me and worried about me being alone in Portugal. Thank you for your support. Thank you to both for always finding time for discussions in your busy schedule.

I would like to thank José Proença, who became an unofficial supervisor and guided me throughout parts of this work as well. Thank you for your patience, for finding time for discussions and for sharing your wisdom.

I would like to thank as well to the academics I met through out these years and contributed with valuable feedback and advice, in particular, to Tomasz Janowski, Marijn Janssen, and the people from UNU-EGOV.

To Mica, who bear with me in the distance and supported me unconditionally. Thank you for making me laugh through all times and for always being present.

To Eduarda and Joana, who became like sisters in Portugal, and will stay with me forever. To Catarina, the other fun half of the "4to esquerdo", thank you for your friendship.

To Lucy and Julia, who I met during my first years in Portugal. Thank you for your friendship and for all the fun we had.

To my brothers and family who I missed very much and with whom I look forward to spend more time.

Last, but not least, I wish to thank my friends from Argentina, whom I miss a lot and skyped with me through the process, and the new I met in Portugal and made my stay more fun: my collogues from the 207 lab, Matias, Lu, the Cecis, Lula, Pancho, Lili, Rita and Adriano.

This work was funded by FCT Foundation for Science and Technology, the Portuguese Ministry of Science, Technology and Higher Education, through the Operational Programme for Human Capital (POCH), PhD grant reference PD/BD/52238/2013. In addition, this work was funded by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by FCT, within project POCI-01-0145-FEDER-016826.

Abstract

In the context of smart cities, governments are investing efforts on creating public value through the development of digital public services (DPS) focusing on specific policy areas, such as transport. Main motivations to deliver DPS include reducing administrative burdens and costs, increasing effectiveness and efficiency of government processes, and improving citizens' quality of life through enhanced services and simplified interactions with governments.

To ensure effective planning and design of DPS in a given domain, governments face several challenges, like the need of specialized tools to facilitate the effective planning and the rapid development of DPS, as well as, tools for service integration, affording high development costs, and ensuring DPS conform with laws and regulations. These challenges are exacerbated by the fact that many public administrations develop tailored DPS, disregarding the fact that services share common functionality and business processes.

To address the above challenges, this thesis focuses on leveraging the similarities of DPS and on applying a Software Product Line (SPL) approach combined with formal methods techniques for specifying service models and verifying their behavioural properties. In particular, the proposed solution introduces the concept of a virtual factory for the planning and rapid development of DPS in a given smart city domain. The virtual factory comprises a framework including software tools, guidelines, practices, models, and other artefacts to assist engineers to automate and make more efficient the development of a family of DPS.

In this work the virtual factory is populated with tools for government officials and software developers to plan and design smart mobility services, and to rapidly model DPS relying on SPLs and components-base development techniques.

Specific contributions of the thesis include: 1) the concept of virtual factory; 2) a taxonomy for planning and designing smart mobility services; 3) an ontology to fix a common vocabulary for a specific family of DPS; 4) a compositional formalism to model SPLs, to serve as a specification language for DPS; and 5) a variable semantics for a coordination language to simplify coordination of services in the context of SPLs.

Resumo

No contexto das cidades inteligentes, os governos investem esforços na criação de valor público através do desenvolvimento de serviços públicos digitais (DPS), concentrando-se em áreas políticas específicas, como os transportes. As principais motivações para entregar o DPS incluem a redução de custos administrativos, o aumento da eficácia dos processos do governo e a melhoria da qualidade de vida dos cidadãos através de serviços melhorados e interações simplificadas com os governos.

Para garantir um planeamento efetivo do DPS num determinado domínio, os governos enfrentam vários desafios, como a necessidade de ferramentas especializadas para facilitar o planeamento eficaz e o rápido desenvolvimento do DPS, bem como ferramentas para integração de DPS, reduzindo altos custos de desenvolvimento e garantindo que os DPS estejam em conformidade com as leis e regulamentos.

Esses desafios são exacerbados pelo fato de que muitas administrações públicas desenvolvem o DPS sob medida, desconsiderando o fato de que os serviços compartilham funcionalidade e processos de negócios comuns.

Para enfrentar os desafios, esta tese concentra-se em aproveitar as semelhanças dos DPS aplicando uma abordagem de Software Product Lines (SPL) combinada com métodos formais para especificar modelos de DPS e verificar propriedades. Em particular, introduz o conceito de uma fábrica virtual (VF) para o planeamento e desenvolvimento rápido de DPS num domínio de cidade inteligente. A VF compreende ferramentas de software, diretrizes, modelos e outros artefatos para auxiliar os engenheiros a automatizar e tornar mais eficiente o desenvolvimento de uma família de DPS.

Neste trabalho, a VF é preenchida com ferramentas para várias partes para planear e projetar serviços de mobilidade inteligente (MI), e modelar rapidamente o DPS com base em SPLs e técnicas de desenvolvimento baseadas em componentes.

Contribuições específicas da tese incluem: 1) o conceito de VF; 2) uma taxonomia para planear serviços de MI; 3) uma ontologia para fixar um vocabulário comum para uma família específica de DPS; 4) um formalismo composicional para modelar SPLs, e servir como uma linguagem de especificação para DPS; e 5) uma semântica variável para uma linguagem de coordenação para simplificar a coordenação.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

Information and Communication Technologies (ICTs) are becoming ubiquitous and immersed in our daily environment contributing to automate and facilitate our activities. They cover a broad spectrum – from software and hardware capabilities, such as wireless networks and mobile computing allowing access to information and services on the move; to the innovative use of existing technologies to enhance public services, such as closed circuit television and pattern recognition used for traffic control.

Due to the embeddedness of ICTs in daily issues, citizens are more and more used to interact with ICTs, putting pressure on governments to take advantage of such technologies to provide better digital public services (DPS). Actually, cities possess a wide range of digital skilled users that are ready to use and benefit from the usage of ICTs to deliver digital public services. In particular, citizens can benefit from the use of ICTs in three main areas: 1) improvements in the way they receive services; 2) enhanced interactions with government; and 3) better quality of lives due to easy access to services. The use of ICTs to provide digital public services also benefits governments by: 1) providing tools to facilitate information sharing among government agencies; 2) increasing effectiveness and efficiency; and 3) providing tools to deliver public value.

1.2 Research Problem

Digital government (e-government) and smart cities provide us a relevant context as well as the motivation for identifying a research problem. In the following we explain this in detail.

Digital government deals with the use of ICTs to facilitate the delivery of digital

public services and support the interaction between their providers and consumers. Since the concept of e-Government was born it has evolved through various stages along the years [86], and is currently evolving towards more specialization and contextualization. In particular, the trend is to concentrate on digital public services focused on a specific policy area – like transport, mobility, social inclusion, or other; or on context-specific conditions – problems affecting a given city. Because of this, e-government rely on technological and organizational knowledge and capacities to ensure effective planning and design of public services. In particular, government officials need specialized tools to understand a given domain – the type of services that can be delivered, the technologies used to deliver them, and the benefits they provide, among others.

This leads to the definition of the first objective of this research work.

O1) To provide a framework for domain-experts to plan and design services in a specific smart city dimension considering context-specific needs

We focus on the smart mobility dimension of a smart city. In particular, smart mobility deals with the use of integrated ICT infrastructures, sustainable transport systems and logistics to support better urban traffic and mobility. Some examples of smart mobility services include the provision of real time and multi-modal public transport services, and traffic light optimization to attend to real-time traffic demand. To address this objective, we proposed the following research questions.

RQ1) What kind of smart mobility services are delivered in the context of smart cities?

RQ2) How are such services delivered?

RQ3) What kind of public value is delivered by smart mobility services and to whom?

In addition, efficient and effective delivery of digital public services encompasses many challenges, including: 1) *rapid development* – to attend increasing citizens' demands and quickly integrate changes in regulations, government must find mechanisms to rapidly develop digital public services; 2) *service integration* – government agencies, and other entities, must collaborate to deliver *seamless* services, i.e. services delivered collaboratively by several government and non-government organizations while presenting a single-organization interface to customers. The *only-once principle* [64], which means that citizens, businesses, and other stakeholders, are required to provide common information only once to government, is particularly critical and requires service integration; 3) *conformance with laws and regulations* – the delivery of services generally depends on laws and regulations, thus government must have mechanisms to ensure

that a digital public service conforms with such laws and regulations; otherwise, failing to correctly design and implement digital public services can increase bureaucracy, or malfunctioning of services; and 4) *development costs* – the adoption of ICT for the development of digital public services involves high costs for governments, which can be difficult to accommodate, particularly at the city level.

In practice, many public services still rely on paper-based solutions, particularly in less resourceful governments. In many other cases, due to the differences in government regulations, lack of interoperability, budgetary resources, and difficulties in ensuring the fulfilment of their specific features, local governments develop tailored ICT solutions to automate the provision of services. This *silo-based* approach exacerbates the aforementioned challenges by increasing development times and costs, as well as by hindering service integration [2]. In addition, it disregards the fact that many public services share common functionality and business processes, not taking advantage of software engineering techniques to efficiently develop *families of services*. By families of services we refer to similar services that share many common features, such as functionality and business processes supporting the delivery of such services, but differ in other features.

Software Product Lines (SPLs) are an efficient approach to develop families of services. Since services must conform with laws and regulations, and exhibit safety and functional correctness requirements, the use of formal methods to formally define an SPL for a family of digital public services seems a feasible approach. Formal methods help to model and verify that a set of services satisfy a given set of properties. In the case of SPLs, they help to verify that the entire family, as well as individual services, satisfy a pre-determined set of properties.

Essentially, an SPL is a set of software systems that share a high number of *features* while differing in others, where concrete products, such as models and systems, are derived from a core of common assets in a prescribed way [45]. In this context, a feature constitutes a functional characteristic or a behaviour of the system visible to the user. The variability of an SPL is defined in terms of common and variable features, usually through *feature models* [7]. A feature model expresses the valid combination of features, where each combination is a product in the family.

There are various formalism to model SPLs, mainly based on automata theory or Petri nets [36, 47, 115]. In the literature, these formalisms are broadly classified into two categories. They can be *annotative* – where all products are specified in a single model. Parts of the model are annotated with variability specifying in which products each part is present. When selecting a set of features only parts associated to those features remain in the model; and *compositional* – where the SPL is modelled in a modular way, where each feature is modelled in isolation, specifying how it alters the core model, i.e. the part common to all products. When selecting a set of features,

only their corresponding models are composed into the final product. In this sense, this approach enables the composition of products.

However, we argue that these approaches differ mainly in the level of granularity of the assets annotated with variability. We recognize a third approach from the literature [115], a truly compositional approach, where the SPL is defined in a modularized way, where each module has its own variability model, which can be composed into a single model. Thus, this approach composes SPLs instead of products. This is of interest because it gathers the best of both approaches documented in the literature. The software engineering problem addressed in this thesis focuses on defining a truly compositional formalism to define SPLs for a given e-government domain, in particular, for public transport services in the context of smart cities.

In addition, given the fact that public services are collaboratively delivered by multiple government agencies, there is a need to orchestrate different services and design how they are integrated and interact. In this sense, we recognize *exogenous coordination* as a suitable approach to orchestrate how variable services, or variable functionality defined in smaller components can be integrated and interact to deliver higher-level functionality. In exogenous coordination, the models of the coordination protocols are separated from the computational models of the components they coordinate. The $\mathcal{R}$eo [9] language is a well-known exogenous coordination language enabling the coordination of components through their interfaces. However, in the case of SPLs, the components to be coordinated have variable interfaces in the sense that they are not present in every product. Thus, we identified the challenge to provide some kind of variable semantics to $\mathcal{R}$eo, such that the coordination protocols used to coordinate variable components can automatically adapt in the presence or absence of those interfaces.

This leads to the definition of the second objective of this research work.

O2) To provide state-of-the art tools to formally specify families of digital public services in a compositional way enabling verification of behavioural requirements

To address this objective we proposed the following research questions.

RQ4) Which modelling technologies are suitable for specifying common features of a family of digital public services delivered by local governments in the context of smart cities?

RQ5) Based on such modelling techniques, how to provide a domain-specific framework, including modelling tools that can automatically generate behavioural models for the members of the identified family of digital public services?

With the aim of packaging the solutions proposed to address each of the research questions described above, and to deliver concrete value to policy makers, we define the last objective.

O3) To identify a family of digital public services delivered by local governments and to provide a repository of models using the proposed framework

To address this objective we propose the following research question.

RQ6) Which family of digital public services in the context of smart mobility is amendable to be deliver through common business processes by local governments in different contextual conditions?

1.3 Solution Approach

The general aim of this research work is to provide conceptual tools for both, government officials and software developers, and rapidly plan and design integrated smart city services on a specific domain of a smart city. Thus, aligned with the objectives and research questions described above we propose the following approach.

In order to address research questions RQ1, RQ2, RQ3 and RQ6 the first part of this thesis investigates smart mobility services and proposes a taxonomy. Taxonomies enable the identification and definition of common concepts of the domain, and layout their relationships, providing a common vocabulary to discuss and share information about the specific domain. Thus, it provides a specialized and contextualized tool for policy makers involved in the development of smart mobility initiatives.

In order to address research questions RQ4 and RQ5 we propose an approach, taking the form of a virtual factory for smart city service integration, for the development and integration of city-level DPS. The virtual factory consists of a framework with tools, guidelines, models, and other artefacts to assist different stakeholders to automate and make more efficient the development of families of DPS. In particular, the virtual factory contributes to different stages in the development of a family of services, namely planning, domain engineering, and software engineering. The thesis populates the virtual factory with elements contributing to the different stages:

1. *Planning* – comprising tools for strategic planning and design of services in a concrete smart city domain;

2. *Domain engineering*, fixing the general vocabulary, attributes, properties, processes, architectural schemes of the selected family of digital public service.

3. *Software engineering*, comprising *formalisms and tools* for rapidly modelling and verifying service properties in the specified domain.

In particular, the software engineering is populated with the following formalism and tools:

a) A *specification language* for service modelling and assembly by feature composition;

b) A *proof-of-concept prototype* to specify, compose, visualize, and translate the relevant models to other well-known formalisms;

c) A *verification engine* to check whether properties documenting the family are satisfied by the models.

In addition, in order to illustrate the formalism and tools, the thesis proposes models and properties of the selected family:

1. A *feature model* specifying domain variability in terms of common and optional functionality

2. *Behavioural models* characterising features representing functionality of the domain, and *temporal properties* of the services to be satisfied by the models

While most of the adopted approaches in the literature take advantages of existing tools, the virtual factory approach proposed in this thesis, discussed in Chapter 8, seeks to help public administrations to move from the silo-based approach of modelling government services to a component-based approach instead. In particular, by identifying and taking advantage of common features and business processes present on different services of a family, and contributing to service integration and interoperability between government agencies.

1.4 Contributions

The work conducted through this research to address the aforementioned domain and software engineering problems, following the proposed approach above, produced the following contributions.

A taxonomy for planning and designing smart mobility services

The taxonomy comprises eight dimensions: type of services, level of maturity, type of users, applied technologies, delivery channels, benefits, beneficiaries, and common functionality. Based on literature review, for each dimension, we synthesize common concepts, provide definitions and illustrate them with the case studies. The taxonomy can assist policy makers to define smart mobility strategies, since it enables the identification of stakeholders to whom services need to be provided, exemplifies the different type of services to be delivered, and the corresponding benefits and beneficiaries,

facilitating the justification of business cases for each initiative. In addition, software engineers can benefit from the identification of common functionality that can be used to develop reusable components for smart mobility services. This contribution is presented in Chapter 3.

A compositional formalism for modelling software product lines

The thesis proposes a compositional formalism, Interface Featured Timed Automata (IFTA), to model in a compositional way the behaviour of SPLs with time requirements, a feature enforced by the presence of recurrent time constraints in public services. We base this formalism on Featured Timed Automata [47], an annotative approach to model SPLs in a single model and annotating it with variability. The main contribution of IFTA is the ability to model SPLs in an incremental and modular way. Each IFTA defines a component that encapsulates some functionality and provides variable interfaces to interact with other components. Each component has its own variability model. When two components are composed, their variability models are composed as well, determining a new family of products. This allows the verification of properties of the composed model against an expected feature model and determine, for example, whether the composed model derives the expected products and only those. We define an equivalence relation between IFTA in order to study properties of the formalism. This relation is based on a bisimulation relation between automata, which can be defined over the entire family or product by product. In addition, we propose a refinement relation for IFTA, in order to verify if a more concrete model of an SPL is congruent with a given specification. Again this relation can be defined over the entire family or product by product. This contribution is presented in Chapter 5 and Chapter 6.

A variable semantics for $\mathcal{R}$eo

$\mathcal{R}$eo is an exogenous coordination language used to orchestrate how distributed components interact. Because in SPL components have interfaces that are not always present in every product, coordination protocols need to adapt to the variable interfaces. Manually defining the variability of a protocol taking into account the variability of each specific component it coordinates is error prone and inefficient. Thus, we propose a variable semantics for $\mathcal{R}$eo using IFTA. Compositionality of IFTA enables the definition of generic coordination protocols that, when composed with other components, adapt well to the components' variability. In addition, the exogenous nature of $\mathcal{R}$eo presents an advantage since it increases the reusability of both the coordination protocols and the components they orchestrate. This contribution is presented in

Chapter 7.

An ontology for licensing public transport services

Most smart mobility services identified during the definition of the taxonomy are either delivered by the private sector or co-created between government and other non-government entities. However, government must ensure the provision of transport as a basic service for residents and regulate such provision. This is done trough the issue of licenses permits, e.g. to provide bus services and for vehicles to transport passengers. This kind of licensing services are offered by most local governments and share many business processes and structural properties. Thus, it is of interest to use this family of services as a case study due to their potential scalability. We study how public transport licenses are delivered in two specific countries – Ireland and Portugal, and propose an ontology to define a common vocabulary for a family of licensing public transport services. The ontology identifies actors, supporting documents, and attributes required in the application and processing stage of the licenses. The main aim of the ontology is to document the structural elements present in the family. However, by defining a common vocabulary, the ontology can serve to: 1) facilitate the transition from paper-based delivery channels to electronic ones; 2) facilitate the integration of different licensing systems, and 3) improving government interoperability. All such features, facilitate information sharing between agencies enabling the delivery of one-stop, seamless services, and the implementation of the "only-once" principle for reducing administrative burden. This contribution is presented in Chapter 8 as part of the domain component of the virtual factory.

A conceptual framework for rapidly modelling families of services.

The proposed framework contributes to the planning, domain engineering, and software engineering stage in the development of a family of services. Contributions of this thesis populate the virtual factory as follows. The taxonomy provides a tool for the strategic planning and design of smart mobility services. The ontology of licensing services contributes to the domain engineering stage by capturing and fixing common vocabulary of the domain. The formalism and tools used for the rapid modelling and verification of families of services, and the concrete models for the selected family contribute to the software engineering stage. The methods and tools used comprise the compositional formalism proposed, IFTA; and a proof-of-concept prototype to model IFTA, compose and translate them to another well-known formalism to reason about behavioural properties, in particular, using the Uppaal model checker. The concept of the virtual factory is configurable by a given family of services, but scalable and

generalizable to other families. This contribution is presented in Chapter 8.

The above contributions produce the following publications.

1. G. Cledou. A virtual factory for smart city service integration. In *Proceedings of the 8th International Conference on Theory and Practice of Electronic Governance*, ICEGOV '14, pages 536–539, New York, NY, USA, 2014. ACM [38]

2. G. Cledou and L. S. Barbosa. An ontology for licensing public transport services. In *Proceedings of the 9th International Conference on Theory and Practice of Electronic Governance*, ICEGOV '15-16, pages 230–239, New York, NY, USA, 2016. ACM [40]

3. G. Cledou and L. S. Barbosa. Modeling families of public licensing services: A case study. In *Proceedings of the 5th International FME Workshop on Formal Methods in Software Engineering*, FormaliSE '17, pages 37–43, Piscataway, NJ, USA, 2017. IEEE Press [41]

4. G. Cledou, J. Proença, and L. Soares Barbosa. Composing families of timed automata. In M. Dastani and M. Sirjani, editors, *Fundamentals of Software Engineering*, pages 51–66, Cham, 2017. Springer International Publishing [44]

5. G. Cledou, J. Proença, and L. S. Barbosa. A refinement relation for families of timed automata. In S. Cavalheiro and J. Fiadeiro, editors, *Formal Methods: Foundations and Applications*, pages 161–178, Cham, 2017. Springer International Publishing [43]

6. G. Cledou, E. Estevez, and L. S. Barbosa. A taxonomy for planning and designing smart mobility services. *Government Information Quarterly*, 35(1):61 – 76, 2018. Internet Plus Government: Advancement of Networking Technology and Evolution of the Public Sector [42]

1.5 Thesis Structure

The rest of this thesis is structured as follows:

Chapter 2 – Domain Background. The chapter describes the role of digital government in delivering digital public services including the challenges faced. Thereafter, it reviews the concept of smart city and its dimensions. Finally, it discusses the smart mobility dimension, presenting the state of the art for smart mobility services in smart cities.

Chapter 3 – A Taxonomy of Smart Mobility Services. First, the chapter briefly describes some background concepts of taxonomy development and the methodology used to build the taxonomy of smart mobility services. Then, it presents a taxonomy of smart mobility services by discussing each of its dimensions and exemplifying with services from the literature. Third, it proposes how the taxonomy can be maintained and by whom, and discusses its validation. Finally, the chapter discusses challenges and lessons learnt during the study, applications of the taxonomy, and concludes with some related work.

Chapter 4 – Technical Background. The chapter introduces SPLs and some of its models, the notion of variability, and discusses how SPLs can contribute to the delivery of digital public services. Then, it explains Featured Timed Automata (FTA), a formalism to model SPLs in which we based the compositional method proposed in the thesis (Chapter 5). Finally, the chapter reviews $\mathcal{R}$eo, a coordination language and the benefits of using exogenous coordination in SPLs modelling.

Chapter 5 – Compositional Modelling of SPLs. The chapter starts by motivating for the need of a compositional formalism to compose and coordinate variable services. Following we present Interface Featured Timed Automata (IFTA), an extension to FTA with variable interfaces and a compositional semantics, discussing its syntax and semantics, and a notion of equivalence in order to study properties of the composition. Finally, the chapter discusses some advantages and limitations of the approach.

Chapter 6 – Refinement of IFTA. This chapter extends the previous one by introducing a refinement relation for IFTA. First it discusses a relation that separates the notion of behavioural refinement from variability refinement, i.e. a product by product refinement relation; and then it proposes a refinement relation over the entire family. Finally, the chapter discusses some decisions made and limitations of the approach.

Chapter 7 – Variability and Coordination. The chapter proposes two approaches to model the $\mathcal{R}$eo coordination language with IFTA, providing the coordination language with a variable semantics. Then, it exemplifies how complex variable coordination protocols can be defined by composing simpler ones modelled as IFTA. Finally, the chapter discusses some advantages and limitations of these models.

Chapter 8 – A Virtual Factory Approach. First, it introduces the concept of the virtual factory. Then, it discusses how the virtual factory was populated in this thesis,

contributing to the planning, domain engineering and software engineering stage in the development process of a family of services. Finally, it discusses some advantages and limitations.

Chapter 9 – Conclusions and Future Work. Finally, the thesis concludes by discussing the research conducted, the advantages and limitations of the contributions, and outlines some possible future work.

Chapters 3 and 5 to 8 contain the original contributions of the thesis.

Chapter 2

Domain Background

This chapter presents background concepts used in the thesis. We introduce the definition of Digital Government and its role in delivering services. We explain benefits that both citizens and government receive from digital public services. We discuss the challenges governments face in order to achieve such benefits. As one of the latest trends in Digital Government, particularly of concern to local governments, we present the concept of smart cities. We introduce six dimensions for the development of smart cities and we focus on the smart mobility dimension. We outline the current state of the art of smart mobility services, which serves as basis for the taxonomy proposed in Chapter 3.

Chapter Organization. Sections 2.1 and 2.2 present the concepts of digital government and digital public services, respectively. Section 2.3 discusses smart cities, and Section 2.4 focuses on smart mobility and summarizes the current state of the art.

2.1 Digital Government

Digital Government deals with the use of ICTs to create public value, relying on an ecosystem comprised by government actors, non-governmental organizations, businesses, citizens' associations, and individuals. The members of the ecosystem contribute with the production of and access to data, services, and content through interaction with the government [122].

According to [86], in the beginning the aim of digital government was to incorporate the use of ICTs to deliver digital public services to citizens. As the concept evolved, the focus on the use of ITCs changed to improve internal processes and efficiency, facilitate collaboration between agencies, and share information; and eventually evolved to en-

gage citizens in government decision, e.g. through the use of electronic participation. Recently, the last stage in digital government evolution suggest the focus is moving towards more specialization and contextualization. This latter stage involves efforts supporting the development of digital public services focused on a specific policy area – such as transport, culture, social inclusion, and economical development, among others; or on context-specific conditions – problems affecting a given city, country, cultural background, or other issues.

2.2 Digital Public Services

A public service is a service delivered for the benefit of the public, especially provided by a non-profit or government organization. The public includes citizens, businesses, and visitors, among others. Examples of public services include notifying, assessing and accepting tax declarations; issuing building permissions and public transport licenses; and providing job-search, among others.

A digital public service is a service that uses ICTs to support the interaction between the service providers and consumers. Examples of interactions includes submitting applications and asking appointments. Examples of digital public services include income tax declarations, notification and assessment; and birth and marriage certificate, as well as another kinds of permissions and licenses' requests and delivery.

From a government perspective, one of the basic responsibilities of local governments is improving citizens' lives by ensuring educational, health, security and transport services, among others. This is ultimately related to improving public services, for example by enhancing accessibility, delivering integrated services and ensuring simplification of service delivery, among others.

Governments must comply with several requirements in order to deliver digital public services. First, services must be provided in a *seamless way*. According to [61] a seamless service "is a public service accessed through a one-stop contact and delivered collaboratively by several government and non-government organizations while presenting a single-organization interface to customers". Thus, citizens do not need to be familiarized with the government structure behind, nor need to interact with different agencies. The different agencies must interact with each other sharing information and coordinating the provision of services, which leads to the concept of "only-once" [64]. This means that citizens should not be asked to give the same information to different agencies more than once. To avoid this, agencies should collaborate to share the information already available.

To deliver seamless services, government agencies and other entities, public or private, must collaborate among them. Such collaboration is related to information

and process integration [61]. Both, information and process integration is a complex task given the nature of the entities involved, in terms of data standards, data semantics, technical infrastructure, software applications, and organizational structures, among others. Most entities use systems that were designed in isolation, and which are not able to easily integrate with others. The maturity level of technological facilities is likely to be diverse among the different entities [55]. It will be also necessary to do changes in the hierarchical responsibilities and duties of the entities if they are intended to collaborate, since most government agencies are fragmented in a way that are independent from each other [150].

According to [61] one key aspect to resolve such issues is interoperability. This is, the capability of government organizations to share and integrate information and business processes based on common standards [73]. Interoperability must be achieved at three different levels in order to provide seamless services [65]: 1) technical – deals with the actual connection and integration between the different software systems and services, including open interfaces, data sharing, data presentation, security and privacy of the data, etc.; 2) semantic – ensures that the meaning of the information exchanged between two systems is preserved and precisely understood; and 3) organizational – defines common business goals and ensures the successful execution of business processes while managing the exchange of information among different entities that have different internal structures, IT platforms and procedures.

To address the challenges faced for improving public services and delivering seamless services, several approaches have been proposed in the literature [49, 88], such as frameworks for digital public service development, interoperability frameworks, enterprise architectures, ontologies, etc. However, most of the approaches proposed take advantage of already developed services and build integrated solutions on top of such services. What we propose in this thesis is to proactively develop families of services by taking advantage of the similarities among business processes and functionality.

2.2.1 Benefits

The delivery of digital public services can benefit citizens and governments. The literature documents many of such benefits, as follows.

Benefits of digital public services for citizens can be classified into three areas:

– *improvements in the way citizens receive services* – e.g., reduction of time spent in bureaucracy by having 24/7 access to government services, one-stop access to government services, and services customized according to citizens needs by using electronic channels and ICT-driven innovations to deliver public services;

- *enhanced interactions between citizens and government* – e.g., enhancing transparency in how services are delivered, by having access to who applied and received public services and the criteria applied for delivering them, enhancing efficiency by eliminating unnecessary tasks and having a collaborative network of public authorities delivering seamless services, and enabling citizens to receive services through digital channels; and

- *better quality of citizens' lives* – e.g., avoiding citizens to queue for hours to receive a public service by enabling access through digital channels, enhanced safety for citizens by improving traffic control or monitoring of public places, greener and less polluted environment by the usage of electric cars, better health services by having access to accurate medical records, and through their consolidation and analysis to more informed health-related policies, etc.

In reality, benefits delivered to citizens are interlinked; for example, having access to integrated and green public transportation systems, improves citizens' quality of life by reducing waiting times and reducing air pollution; having better access to government information makes citizens more aware of governments issues and encourages them to participate, which in turn strengthens democracy and makes governments more transparent. This interlinking is better illustrated in Section 3.3.6 when we discuss the benefits identified from the delivery of smart mobility services, and classify them by the smart city dimension to which they contribute.

Benefits of digital public service for government include:

- *facilitating information sharing among government agencies* – by digitalizing information and making use of communication platforms [62];

- *increasing effectiveness and efficiency* – by designing common service components lie authentication, payment and notification services, and integrating systems from different domains such as transportation, health and environment [93]. The automation and standardization of public services leads to reducing bureaucracy [55]; and

- *embedding ICTs in daily activities and obtaining public value through it* – making use of ICT-innovations to automate processes and deliver ubiquitous and seamless services, e.g. by using RFID devices embedded in cars in combination with sensors and CCTV systems to automatically detect and generate fines for excess of speed, illegal parking, etc. [20]

2.2.2 Challenges

When developing digital public services, local governments face several types of challenges. In particular, some of the technical challenges include the possibility of identifying common functionality existing in the business processes of public services and the scarcity of domain-specific reliable tools enabling the automated composition of common functionality for the rapid development of digital public service.

The wide range and complexity of services to be provided, in addition to the fact that systems are usually built following a "silo-based" approach and designed independently from others, hinders service integration and collaboration between government agencies as well as between public and private organizations [2]. In addition, the lack of resources in many local government, contributes to the fact that many of services still relay on paper-based solutions. The lack of tools for overcoming such challenges sets back the development of efficient and effective digital public services which can deliver real value to citizens.

Examples of technical and organizational challenges for digital public service development include the following.

Technical Challenges

- *rapid development of digital public services:* to attend increasing citizens' demands and quickly integrate changes in regulations, government must find mechanisms to rapidly develop digital public services [87].
- *service integration:* government agencies, and other entities, must collaborate to deliver *seamless* services [62].
- *multichannel delivery:* governments need to enable service delivery using traditional (e.g., counter, phone) and digital channels (e.g., website, mobile devices) for service delivery since Internet may not be accessible for the whole population [55]; for example due to the lack of digital skills, and lack of motivation exhibited by service recipients to use Internet.
- *concerns about privacy and security:* some citizens may not be willing to consume digital public services because they feel their privacy and security at risk. Digital public services and most important information integration processes performed for delivering one-stop services must guarantee the privacy and security of personal data [150].
- *usability:* the acceptance of digital public services is highly affected by the friendliness, easy of use, reliability and other qualities of the software applications supporting them [56].

Organizational Challenges

- *business process integration:* the integration of back office processes is usually complex given the diverse nature of tasks performed, organizational structures executing them and ICT solutions used [55]. Frequently, it involves re-engineering current processes which in turn may require significant organizational changes in terms of agencies structures and responsibilities [150].
- *matching citizens needs:* instead of using a qualitative approach, governments tend to follow a quantitative approach to develop digital public services. Following this latter approach, services are delivered without analysing what citizens really need or the way they would like to receive public services [56].
- *development costs:* the adoption of ICTs for digital service delivery usually involves high costs for local governments [116].
- *conformance with laws and regulations:* the delivery of services generally depends on laws and regulations. Government must have mechanisms to ensure that an DPS conforms with such laws and regulations, otherwise, failing to correctly design and implement DPSs can increase bureaucracy, and result in unused or malfunctioning of services [150]

2.3 Smart Cities

Various definitions of the smart city concept exist in the literature [76, 78, 149]. For example, Harrison et al. [78], offer a comprehensive and simple definition of smart city: "an instrumented, interconnected, and intelligent city".

In the context of the above definition, *instrumented* refers to the capacity to acquire live real city data through various channels sensors, meters, safety cameras, personal devices, satellite navigation systems, social networks, etc. *Interconnected* refers to the mechanisms for data integration and their usage to deliver enhanced city services. *Intelligent* refers to the use of advanced computational tools to analyze collected data, obtaining valuable information to assist in better government decision-making processes and in delivering public value to citizens and society at large. In accordance with such definition, a main objective of smart city development is to provide government officials and citizens with real time, valuable information in order to improve governments performance and citizens quality of lives.

To be considered smart, a city needs to make strategic use of ICTs to achieve significant improvements across different city dimensions. Giffinger et al. [70] recognize

six dimensions: smart economy, smart environment, smart governance, smart living, smart people, and smart mobility.

In particular, smart economy refers to city competitiveness, including innovation, entrepreneurship, and productivity. Smart environment concerns issues related to natural resources and attractive natural conditions of the city, like green spaces, climate, and green practices. Smart governance considers citizens participation in decision-making processes; public and social services delivered to citizens, visitors and other city stakeholders; as well as government administrative processes. Smart living focuses on issues related to quality of life, including culture, health, housing, education, and tourism. Smart people addresses issues related to human and social capital, like citizens level of education or qualification and quality of social interactions. Finally, Smart mobility focuses on the use of reliable and integrated ICT infrastructures, sustainable transport systems and logistics in support of better urban traffic and inhabitants mobility.

2.4 Smart Mobility Services

By 2050 it is expected that 66% of the world population will reside in urban areas [146]. As the number of urban residents increases, local governments need to address serious sustainable and development challenges in various areas, including mobility. Mobility issues impact on citizens quality of life and the overall sustainability of cities. For example, travel time shows a strong positive relationship with life satisfaction in smaller cities, but such relation is non-existent in large cities, mainly due to the costs of traffic congestion [112]. Regarding sustainability, in the United States, transportation is responsible for 27% of the greenhouse gas emissions [147], while in developing countries the transport sector is responsible for 80% of air pollution [145]. Globally, it is estimated that road transport consumes about 70% of the energy used in the world transport system and that only road passenger transport accounts for 50% of this energy consumption [57]. Additionally, as part of the Sustainable Developments Goals (SDGs)[1], the goal number 11 (SDG11) refers to make cities more inclusive and sustainable. In particular, target 11.2 defines that by 2030, governments should provide access to safe, affordable and sustainable transport systems for all.

In addition to their relevance, the planning and development of smart mobility services is challenging. One of the challenges is that digitization policies and strategies need to carefully consider the interests and needs of the many stakeholders involved (government, citizens, commuters, transport providers, etc.), such that possible (un)expected negative effects to some group of stakeholders are minimized.

[1] http://www.un.org/sustainabledevelopment/cities/

Existing solutions, examples of good practices, have been implemented in smart cities, offering a catalogue of initiatives from which governments can learn and consider for adoption in their own local context.

However, the information available of such initiatives is shallow, unstructured, and not properly maintained. In addition, given the lack of information- and experience-sharing, each local government develops its own ad-hoc solutions to deliver mobility services, ignoring that in practice many of such services share common functionality and thus, could be built using reusable components simplifying development processes and significantly reducing costs.

With the aim of understanding the domain, and addressing research questions RQ1) to RQ3) and RQ6), we investigate the state of the art in smart mobility services in the context of smart cities. The following two sections outline the state of research and practice of such services, repectively.

2.4.1 State of Research

To understand the state of research on smart mobility services, we conducted literature review using a scientific repository. We selected Scopus[2] as the main source due to its coverage of journal publications and publications on hard sciences, e.g., computer science, relevant to smart city initiatives. Searches were conducted using the keywords "transport or mobility" and "smart city", and "'smart transport' or 'smart mobility"' and "city". We considered papers that refer to ICT solutions and only the most influential ones, i.e. papers with more than two years since their publication with no citations were discarded. In total, 18 papers were selected.

Findings from the state of research assessment include: P1) a software platform to automate the collection and aggregation of large scale context information provided by different sources, which serves to build an intelligent transportation system to better understand traffic problems [59]; P2) an ICT platform for an intelligent transportation system that updates users with real time information regarding flexible transport, like bicycles and car sharing, and traditional transport systems, such as optimal routes, delays in public transport, available parking places, and the state of the roads with data collected from various sources [6]; P3) an intelligent urban traffic management system that detects congestions from various information sources, such as sensors at road intersections and on public transport vehicles [133]; P4) a fuzzy neuronal network to optimize traffic light patterns and provide priority to public buses and emergency vehicles [80]; P5) an adaptive system for intelligent traffic management [72]; P6) a framework for designing business services, and its application to mobility services [114];

[2] https://www.scopus.com/

P7) a cloud-based car parking middleware for smart cities based on Internet of Things, including sensors to detect available parking places, wireless technologies to transmit the information sensed, and functionality to provide cloud-based parking services [90]; P8) a large scale traffic simulation platform for transport authorities to optimize city transportation [139]; P9) a multi-agent simulation system that incorporates real world data into the simulation, e.g. snowstorms, to improve smart transportation planning [19]; P10) a real time mobility assistant providing information about multi-modal journeys [113]; P11) an augmented reality application to access public transport information, such as buses arrival times and routes [128]; P12) a generic framework for mobile participatory sensing with a live transit feed service providing real time information about public transport services [140]; P13) an application layer solution for a geo-casting function for smart on-board units in vehicles. The aim is to automatically detect car accidents and disseminate information about the incidents to near vehicles and authorities, like hospitals and police stations [141]; P14) an assistance system to adjusts the driving speed to arrive at intersections when traffic lights are still green when buses have their own lane such as Rapid Transit Buses [136]; P15) an electrically powered one person transport system for pedestrian areas, such as historical city centres, enabling accessibility for people with mobility disabilities [31]; P16) an adaptive traffic management system with a fuzzy logic scheme to reduce travelling times of emergency vehicles [58]; P17) a platform for dynamic carpooling in the city, i.e. to share car rides that are published in real time, contrary to typical carpooling services that required planning in advanced [109]; and P18) an approach to reduce traveling times suggesting customized routes to vehicles to avoid traffic jams using vehicle-to-infrastructure communication [138].

2.4.2 State of Practice

To assess the state of practice, we conducted searches using Google search engine and to identify smart city good practices, we used the keywords "'smart city' or 'intelligent city' or 'living labs' or 'ciudad inteligente"', in conjunction with three continent names, America, Asia, and Europe, to cover countries with different levels of development. For each city, the following selection criteria was applied: availability of information of mobility services in government websites, and diversity of mobility services with respect to those already collected. In total 9 cities were selected as shown in Table 2.1.

All the nine cities have been recognized as cities standing out in some of the six dimensions of a smart city. In particular, Curitiba is recognized[4] by its integrated transportation system and land use planning [5]; Seattle is a leading example in sus-

[4] https://sustainabledevelopment.un.org/index.php?page=view&type=99&nr=57&menu=1449
[5] http://www.ippuc.org.br

Table 2.1: Smart cities selected.

Continent	City	Country	Reference [3]
America	Curitiba	Brazil	http://www.curitiba.pr.gov.br
	Seattle	USA	https://www.seattle.gov
	Surrey	Canada	https://surrey.ca/city-government/15430.aspx
Asia	Dubai	UAE	http://www.dubai.ae/en/Pages/default.aspx
	Songdo	South Korea	http://songdoibd.com/about/
Europe	Amsterdam	The Netherlands	https://amsterdamsmartcity.com
	Barcelona	Spain	http://smartcity.bcn.cat/es/
	Copenhagen	Denmark	https://stateofgreen.com/en
	Vienna	Austria	https://smartcity.wien.gv.at/site/en/

tainability in terms of energy efficiency and transportation[6], and has recently joined a national network of smart cities [7]; Surrey was recognized as one of the top seven intelligent communities of 2016 [8]; Dubai was recognized as the leading smart city initiative in the Gulf [9]; Songdo is well known because the city was conceived and built from scratch [10]; Amsterdam was recognized with the Smart City World Award in 2012 and is an example in living labs, and mobility and smart grid programs [11]; Barcelona was recognized as World's Smartest City in 2015 because of its programs in smart traffic management and smart grids[12]; Copenhagen is recognized due to the Copenhagen Connecting initiative, awarded as best project with the Smart City World Award in 2014 [13]; and Vienna was recognized with the Smart City World Award in 2016 [14] for its integrated energy research.

Below we summarize examples of the services that are delivered by each city. References to each initiative can be found in [42].

Amsterdam. S01) *Moby Park*, a mobile application and website facilitating finding, booking and paying parking places and getting driving guidance to arrive to a selected parking place. S02) A platform for finding and booking parking places, and for receiving driving guidance for lorry drivers. S03) A mobile application to provide real time

[6] http://smartcitiescouncil.com/article/why-seattle-sustainability-superstar
[7] https://www.seattle.gov/tech/initiatives/smart-cities
[8] https://surrey.ca/city-government/15430.aspx
[9] http://e.huawei.com/ae/news/ae/2015/2016/201610181814
[10] http://songdoibd.com/about/
[11] http://www.smartcityexpo.com/past-editions-2012
[12] https://eu-smartcities.eu/content/barcelona-world's-smartest-city-2015
[13] http://www.smartcityexpo.com/past-editions-2014
[14] http://www.smartcityexpo.com/en/past-editions-2016

driving guidance, using calendar data to suggest when to start driving so to arrive on time to scheduled appointments. S04) A mobile application to provide real time driving guidance to emergency vehicles. S05) *WeGo*, a peer-to-peer car-sharing platform for car owners to share their cars, and for non-car owners to find, book and pay for available cars. It also enables to lock and unlock cars.

Vienna. S06) An energy saving tram that monitors different aspects of the journeys and takes proactive measures to increase passengers' comfort, e.g., it controls the amount of incoming fresh air and sunlight. S07) *Smile*, a mobile app to plan multi-modal journeys, to book and pay mobility services, to open doors at parking places and shared vehicles, and to access mobility records.

Copenhagen. S08) *Copenhagen Connecting*, an integrated system tracking moving assets for security or location purposes, sharing real time data about parking places, and monitoring and controlling traffic flow on real time. It also enables traffic light optimization, and dynamic pricing of parking and road tolls based on current parking demand and traffic flow. S09) A cloud-based dashboard to study the efficiency of traffic light patterns, traffic behaviour, and correlations between traffic and other influencing factors. S10) An integrated system to coordinate traffic lights and provide faster green lights for buses, to detect parking places, to inform passengers about delays and alternative routes, and to widen or shrink lanes on rush hours through dynamic led signs. S11) *Rejseplanen*, a real time multi-modal journey planner with customizable searches, and displays with information about transport. S12) An short messaging service (SMS) to buy public transport tickets.

Barcelona. S13) A remote device for blind people to command traffic lights to emit an audible sound to safely cross the street. S14) An intelligent traffic light system to control traffic lights and to provide faster green lights for emergency vehicles. S15) *ApparkB*, a mobile app to pay street parking places, providing access to a monthly summary of expenses. S16) *App&Town*, a real time multi-modal journey planner for public transport gathering information from various sources. S17) An SMS service to access transport-related information, including availability of public bicycles, and cars seized by authorities. S18) *JoinUp Taxi*, a mobile app to share a taxi, rate taxi drivers and users, estimating money and carbon dioxide (CO2) emissions saved by sharing a taxi. S19) *WeSmartPark*, a mobile app and website to find available parking places, get driving guidance, book places, issue payments when leaving the parking place, and automatically opening parking doors.

Curitiba. S20) Services for public buses including wireless Internet on-board, and smart pass payment, providing real time information about the service; and monitoring vehicles and their performance, e.g., measuring passenger demand to send more vehicles to cover a route, and comparing estimated traveling times with actual times. S21) A service providing online integrated itinerary comprising information about various types of buses. S22) A service estimating traveling times by detecting and monitoring cars entering and leaving specific locations. S23) A system to control traffic lights based on current traffic conditions or by request. S24) A system providing faster green lights for public buses, and monitoring buses to identify critical and optimal road intersections to improve traveling times. S25) A system providing real time traffic information through dynamic message signs. S26) An operational control centre to monitor traffic and incidents in real time through closed-circuit television (CCTV). S27) Radars and cameras to detect vehicles that exceed the speed limit, fail to stop on a red light, or stop on the pedestrian walk.

Songdo. S28) A system for optimizing traffic light aiming at resolving traffic congestions, which are detected by analysing geo-location information provided by radio-frequency identification (RFID) tags on cars.

Seattle. S29) A website providing real time information of available parking places, and a system of dynamic message signs providing guidance to the nearest available parking place . S30) Online payment of parking and traffic fines. S31) An online interactive map with information about planned events that affect mobility. S32) A system to monitor traffic flows and to automatically detect license plates for security and law enforcement, estimating traveling times. S33) A system to adjust traffic lights based on historical and current data, and to provide faster green lights for public buses. S34) A system providing real time information about traffic through dynamic message signs. S35) An online interactive map publishing real time traffic information, including incidents, planned events, and access to CCTV cameras feeds.

Dubai. S36) *Nol Card*, a smart card to seamlessly pay for different mobility services, including public transport and parking places. S37) *Smart Taxi*, a mobile app to request taxis, choose the type of vehicle, find out the location of taxis, get information about the driver, and rate the drivers driving skills. S38) A multi-modal journey planner with customizable searches and estsimations of traveling times and costs. S39) A mobile app to find nearby available parking places and to receive driving guidance to reach the parking place.

Surrey. S40) A mobile application for car-pooling. S41) A website for car-sharing, including a booking service. S42) A system to control traffic lights and to adjust them for faster emergency responses, as well as to guide emergency vehicles, and to analyse traffic behaviour.

Chapter 3

A Taxonomy of Smart Mobility Services

With the aim of addressing the lack of structured information and deepening the knowledge in smart mobility services, as well as to address research questions RQ1, RQ2, and RQ3, we analysed the state of the art described Section 2.4. Based on the analysis and findings, we propose a taxonomy for planning and designing smart mobility services. The taxonomy comprises eight dimensions: 1) type of services, 2) maturity level, 3) type of users, 4) applied technologies, 5) delivery channels, 6) benefits, 7) beneficiaries, and 8) common functionality.

The structuring nature of taxonomies enables to identify and define common concepts for each of the dimensions, providing a common vocabulary to discuss and share information about smart mobility services. In addition, it provides a specialized and contextualized tool for policy makers involved in the development of smart mobility initiatives. In particular, the concrete dimensions identify the spectrum of mobility services that can be provided, to whom they are provided, what technologies can be used to deliver them, and the public value that is delivered through each kind of service. Identifying common functionality can also help software engineers and IT staff to implement smart mobility services through reusable components, ready to be configured and integrated into software applications.

Chapter Organization: Some background on how taxonomies are built is presented in Section 3.1. Section 3.2 describes the methodology used for building the taxonomy. Section 3.3 introduces the proposed taxonomy. Sections 3.4 to 3.6 discuss the validation and maintainability of the taxonomy, and some challenges and lessons learnt, respectively.

3.1 Building Taxonomies

Taxonomy is the science of classification. It structures information of a given domain into groups and lays out their relations, providing a conceptual framework for discussion, analysis, and information retrieval [28]. We use a taxonomy since we are merely concerned with the classification of concepts, although it can be later evolved into an ontology with richer relations and characterization of the concepts. Below we discuss some key aspects of taxonomy development.

3.1.1 Taxonomy Structure

The most common types of relations between concepts are hierarchies, trees, and faceted [101]. We focus on the faceted structure due to its many advantages. The approach considers that there are multiple perspectives or facets to model a concept. Main advantages include: 1) *hospitability* – it does not require a complete knowledge of the domain. This is attractive for emerging or changing domains, as the smart mobility domain, which is continuously evolving due to advances in technology and changing needs; 2) *flexible searches* – facilitates recovering information in multiple ways; e.g., some of the benefits delivered by type of service; 3) *greater expressiveness* – each facet can use the structure that best suits the knowledge that it represents; and 4) *flexibility* – each concept can accommodate multiple perspectives. As a limitation, facets do not explicitly express meaningful relations between concepts.

3.1.2 Taxonomy Development

Categories in a taxonomy are constructed following an iterative process. In each iteration a development approach is selected and at the end of the process it is analysed if categories are well defined, need to be merged, or if new ones can be identified [120].

There are three well known development approaches [18]: 1) conceptual, 2) empirical, and 3) operational. The last one is a combination of the previous two and is the most commonly used in practice.

An operational approach can follow two development patterns. On the one hand, *conceptual to empirical*, where categories are first conceptualized following a deductive process, based on theory, domain knowledge, or experience, and then empirical cases are identified for each concept. On the other hand, *empirical to conceptual*, where a series of empirical cases are first identified, analysed and grouped based on recognized similarities, and then conceptual labels are formulated for them.

In addition, various methodologies and best practices exist for taxonomy develop-

ment. We identify three that are generic enough to easily adapt to our domain [1]: 1) BR [28], and 2) CJ [32] – both focusing on organizational aspects; and 3) NVM [120] – focusing on information systems. We believe that the three methodologies complement each other, and as such, we propose a methodology combining guidance and steps from all of them. In particular, NVM recognizes the need for an iterative development process and provides guidance for selecting a development strategy, the criteria to develop a useful taxonomy and how to use such criteria to evaluate the taxonomy; BR recognises the need of a data collection process; and BR and CJ distinguish different taxonomy structures and provide guidance for maintaining the taxonomy. The proposed methodology is described in Section 3.2

3.2 Methodology

The proposed methodology for taxonomy development comprises five steps: 1) *Planning*, 2) *Data Collection*, 3) *Taxonomy Construction*, 4) *Validation*, and 5) *Maintenance*. A comprehensive view of the methodology is depicted in Figure 3.1, including the five main activities, tasks involved in each activity and some results obtained. The activities conducted and decisions made in each step are described in the following sections.

3.2.1 Planning

This step provides the foundations to develop the taxonomy. It defines the goals and scope of the taxonomy, meta-characteristics, ending conditions, and structure of the taxonomy.

Goals, Scope, and Meta-characteristics. The goal is to structure knowledge related to smart mobility services. The scope of the taxonomy is limited to smart mobility services, in particular, software intensive services, with limited attention given to services depending mainly on non-software technologies, such as electric vehicles. The research questions RQ1, RQ2 and RQ3 formulated in Section 1.2 serve as the meta-characteristics of the taxonomy, i.e. they represent the most comprehensive characteristics that will serve to determine the features of the taxonomy. The main dimensions of the taxonomy should be a logical consequence of these meta-characteristics.

[1] We use the authors initials to identified them i.e. BR, CJ, and NVM.

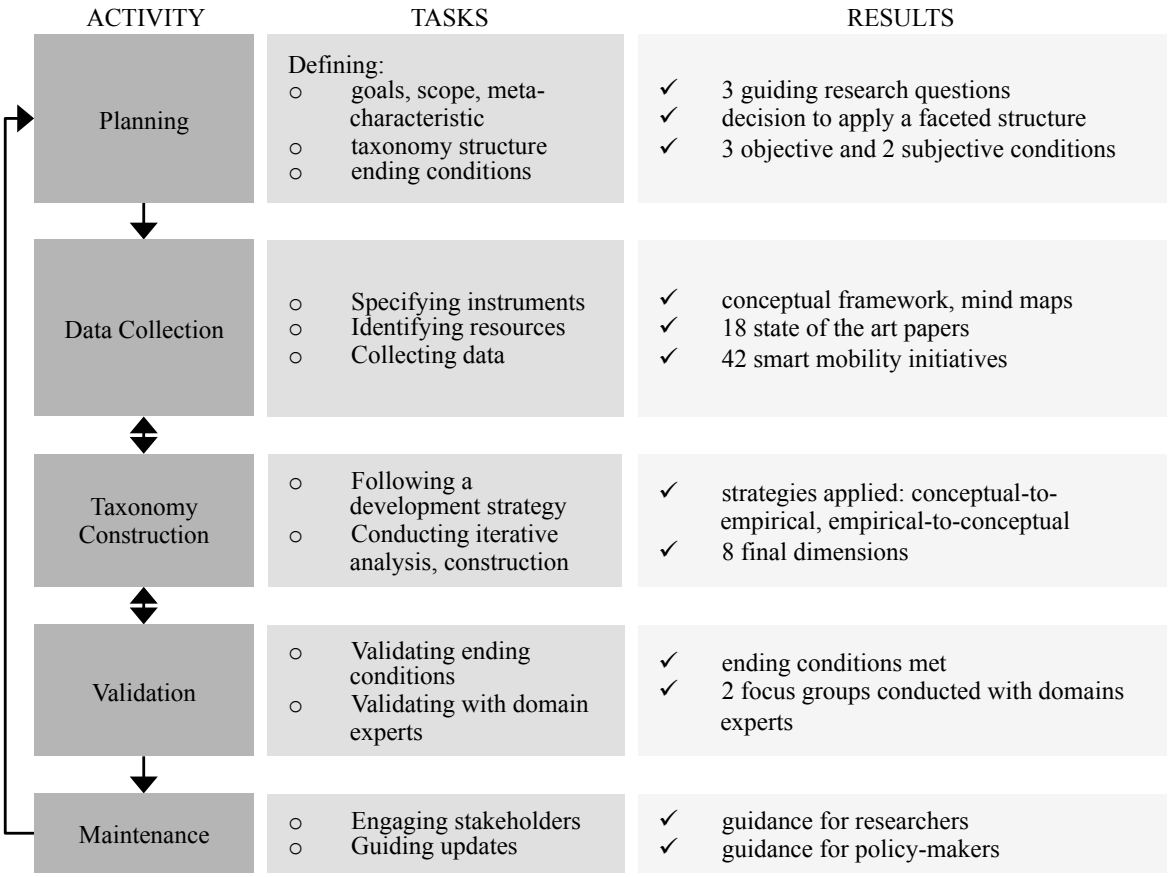


Figure 3.1: Methodology for taxonomy development

Taxonomy Structure. We selected a faceted structure due to its many advantages as discussed in Section 3.1.1, and due to the different aspects, or facets, explored by the meta-characteristics.

Ending Conditions. We selected the following objective and subjective ending conditions, based on NVM. Objective conditions: 1) no dimensions or characteristics were merged or split in the last iteration; 2) no new dimensions or characteristics were added in the last iteration; and 3) each dimension and characteristic is unique and not repeated. Subjective conditions: 1) the number of dimensions allows the taxonomy to be meaningful without being unwieldy or overwhelming; and 2) dimensions and characteristics can be easily added.

3.2.2 Data Collection

This step comprises the identification of resources and instruments to collect data, and the actual collection of data. We divided the data collection process into state of research, to collect data from scientific publications on smart mobility solutions, and state of practice, to collect data about mobility services delivered in the context of smart cities. The results of this step have been presented in Section 2.4, when we discussed the state of the art of smart mobility services.

The data was analysed based on three constructs to understand the domain – What, Who and Why. Each construct aligns with a concrete research question (Section 1.2), as follows:

- What – RQ1): What is the service about? The construct explores the aim and type of the service delivered.
- How – RQ2): How is the service delivered? The construct investigates the technology and the channels used to implement and deliver the service.
- Why – RQ3): Why is the service relevant? The construct assesses for whom the service is delivered and the public value delivered through the service.

3.2.3 Taxonomy Construction

This step is an iterative process. In each iteration, a strategy is selected to build or refine the taxonomy, a series of steps are conducted depending on the selected strategy, and the ending conditions are checked to decide whether another iteration is needed.

First, the top-level dimensions of the taxonomy were conceptualized following a *conceptual to empirical* approach, i.e. for each research question, we identified concepts that capture the essence of the question. In particular, six dimensions were identified at this stage: 1) *type of services* as logical consequences of RQ1; 1) *applied technologies* and 3) *delivery channels*, as logical consequences of RQ2; and 4) *benefits*, 5) *beneficiaries* and 6) *type of users*, as logical consequences of RQ3.

Second, various iterations were conducted following an *empirical to conceptual* approach until the ending conditions were met. In each iteration, services of the initiatives were grouped based on common characteristics under the different dimensions, given rise to the various sub-categories of each dimension. For example, in the type of user dimension, we identified groups of services for people who drive cars and for cyclists. Intuitively, these groups gave rise to the sub-category drivers. This was conducted with the help of conceptual maps to group services and for this we used the XMind[2] mind-

[2] https://www.xmind.net

mapping tool. During this process, it was evident that many services provided similar functionality with small variations, resulting in the identification of another dimension, 7) *common functionality*, which can be seen as a logical consequence of RQ2.

3.2.4 Validation

This step comprises the validation of the taxonomy and the integration of the received feedback. We organized two focus groups meetings with international government practitioners and academic experts with experience on the smart mobility domain.

The two meetings conducted in Guimarães, Portugal, one at Comunidade Intermunicipal do Ave [3] and one at United Nations Universit-EGOV (UNU-EGOV) [4], were organized as two-hour session, including 20-minutes presentation of the taxonomy, followed by a discussion among participants. The aim was to discuss the suitability of the taxonomy, validity of the concepts, completeness, weaknesses, and improvements. In total, the meetings were attended by five government experts from China (working for Beijing Government), Uganda (working for Ministry of ICT, Uganda), Portugal (working for Comunidade Intermunicipal do Ave), and Denmark (former government official of the Danish Agency for Digitization), and four academics from UNU-EGOV.

Both meetings provided valuable feedback for validating the content of the taxonomy as well as for its improvement. In particular, a new dimension was incorporated, 8) *level of maturity*, which can be seen as a logical consequence of RQ2. This resulted in a new iteration of step 3, where the characteristics for this dimension were identified following a *conceptual to empirical* approach based on existing theory.

In addition to the feedback received during the two focus groups, the content of the taxonomy has been validated based on traceability, i.e. the content has been produced only based on collected data and references are provided for each concept.

3.2.5 Maintenance

This step comprises the identification of the stakeholders responsible for the continued maintenance and evolution of the taxonomy, and the definition of guidelines for the stakeholders. These topics are discussed in Section 3.5.

[3] http://www.cim-ave.pt
[4] https://egov.unu.edu

3.3 A taxonomy of smart mobility services

The main dimensions defined for the taxonomy of smart mobility services are depicted in Figure 3.2. The identified values for each dimension are defined and illustrated in the following sections.

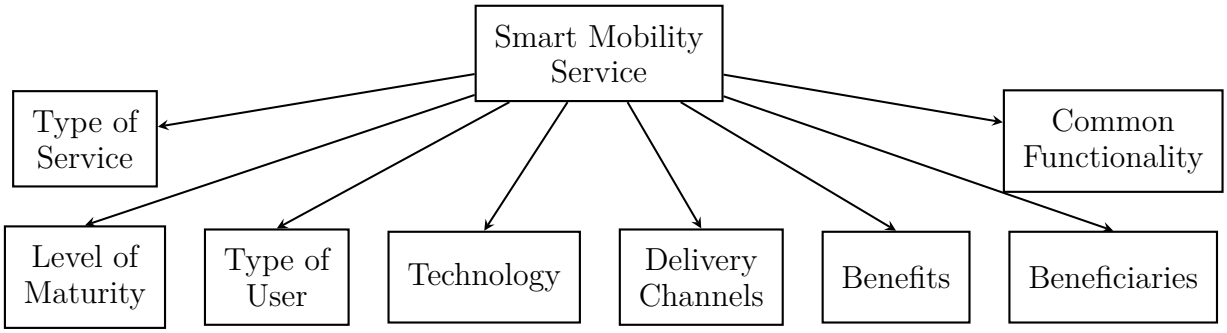


Figure 3.2: Defining a smart mobility service taxonomy

3.3.1 Type of Services

We identify 12 *types of services*. Table 3.1 describes each type and exemplifies it by classifying each initiative described in Sections 2.4.1 and 2.4.2.

Table 3.1: Type of service

Category	**Description**	**Examples**
Driving guidance	It provides guidance to drivers about the best route for moving from one place to another, including fixed or real time information of issues affecting mobility. Routes can be selected based on different criteria, such as the shortest or fastest route.	S01, S02, S03, S04, S19, S39, S42, P18
Improving transport infrastructure	It refers to enhanced functionality included in transport resources, usually related to a specific goal, such as energy savings, enhancing the travel experience, and reducing CO2 emissions.	S06, S20, P13, P14, P15
Improving transport infrastructure	It refers to enhanced functionality delivered to transport infrastructure such as parking places, roads, traffic lights, etc., including devices to detect empty parking places, or dynamic message signs to inform about traffic.	S07, S10, S13, S19, S25, S29, S34

Journey planners	It provides instructions for moving from one place to another using one or multiple types of transport for a single journey. Instructions include types of transport available, travel and arrival times, and guidance for commuting between them. Guidance can be personalized with different criteria such as the cheapest, fastest or most environmentally friendly journey, and support for transporting wheelchairs or bicycles, among others.	S07, S11, S16, S20, S21, S38, P2, P6, P10
Locating objects	It enables to locate, usually in real time, vehicles in the city such as cars, public bicycles or public buses.	S08, S17, S37
Monitoring traffic	It enables authorities to monitor, analyse, and get insights about traffic and pedestrians, such as detecting congestions, estimating travelling times, and detecting cars' illegal behaviour. It can rely on monitoring of simulated data, e.g., to evaluate the impact of events on traffic, such as, weather, road closure, and adjustments in traffic lights patterns. Recorded data can serve as evidence of incidents for authorities.	S08, S22, S26, S27, S32, S42, P1, P2, P3, P5, P8, P9
Monitoring transport	It enables transport authorities to monitor public transport vehicles to get insights about on-board events, vehicles' performance based on current and expected travelling times, number of passengers, and incidents, among others.	S20, S24
Parking	It enables users to search, book, and pay for parking places. It can also include functionality for managing the parking facilities, offering parking places and managing interactions between the parking provider and the users.	S01, S02, S08, S10, S15, S19, S29, S39, P2, P7
Payment	It enables to seamlessly pay transport-related services, such as tickets for single or multi-modal journeys, parking places, energy for electric vehicles, road tolls, and use of public bicycles, among others.	S01, S05, S07, S12, S15, S19, S20, S30, S36
Reporting mobility	It provides various stakeholders with information about events affecting mobility such as planned events, incidents, alternative routes, and current travelling times, among others.	S10, S25, S31, S34, S35, P2, P11-P13
Sharing transport	It enables to share vehicles (car-sharing) and journeys (car-pooling), including, announcing, searching, booking, and paying for cars and shared journeys, and accessing to vehicles.	S05, S18, S40, S41, P17
Traffic light optimization	It allows adjusting traffic light patterns based on different factors including current traffic flow, historical and simulated data, and approaching of special types of vehicles such as emergency vehicles and public buses. The main aim of this type of service is to respond to the changing demands in traffic flow and to prioritize the moving of special vehicles.	S08, S09, S10, S14, S23, S24, S28, S33, S42, P4, P16

3.3.2 Level of Maturity

We define *level of maturity* following the United Nations four-stage digital public service maturity model [104] to classify standard digital public services, and the digital public service innovation framework proposed in [26] to classify innovative digital public services. Table 3.2 describes each level of maturity and classifies each initiative. Example services were identified for all categories.

Table 3.2: Level of maturity

Category	**Description**	**Examples**
	Standard Services	
Emerging	Informational services providing basic online information with only one-way interaction between users and the service provider.	S25, S29
Enhanced	Services with enhanced one-way communication or basic two-way interactions between users and the service provider.	S31
Transactional	Services enabling complete transactions online with two-way communication between users and the service provider.	S13, S17, S30
Connected	Integrated seamless services, delivered using multiple technologies and platforms, and highly interactive.	S01-S05, S07-S12, S14-S16, S18-S21, S24-S27, S29, S32-S42
	Innovative Services	
Transparent	Services publishing information about service context and decisions made by the service provider.	S07, S31, S35, S37
Participatory	Services using participatory techniques such as crowdsourcing.	S16, S18
Anticipatory	Services that anticipate to the needs of the user.	S03
Co-created	Services delivered collaboratively between government, private sector, and/or non-government organizations, and/or citizens.	S01, S05, S18, S40, S41
Personalized	Customizable service delivery based on user's preferences and specific needs.	S07, S11, S16, S18, S38
Context-aware	Services or information provided based on the user's context information.	S01, S02, S04, S07, S10, S20, S39
Context-smart	Services that apply intelligence and leverage on the knowledge of context-related information to take action at the needed moment.	S02-S03, S06, S08-S10, S14, S19, S22-S24, S28, S33

3.3.3 Type of Users

In this context, a *user* refers to any actor receiving a smart mobility service. We identify five main types of users. Table 3.3 describes each category and classifies each initiative.

Table 3.3: Types of users

Category	Description	Examples
Transport Authority	A government agency responsible for the licensing of public and commercial vehicles, for designing, building and maintaining land transport resources and infrastructures, and for regulating and monitoring land transport in a given territory.	S08-S10, S14, S20, S22-S28, S32, S33, S42
Driver	A person who is licensed for driving a vehicle or is able to drive a vehicle that does not require a permit. Drivers are further classified into Motor-Vehicle Driver, and Bicycle Driver.	S01, S02-S04, S05, S07, S08, S10, S15, S17, S19, S25, S29-S31, S34- S36, S39-S41, S42
Passenger	A person who travels in a vehicle without participating in its operation. Passengers are further classified into public and non-public transport, and taxi passenger.	S06, S07, S10, S11, S12, S16, S18, S20, S21, S36-S38, S40
Resource Owner	A person who possesses a transport-related resource, such as a vehicle, parking place, garage, and is willing to share it or rent it.	S01, S05
Pedestrian	A person who walks through the city.	S13, S16, S23

3.3.4 Technology

This concept focuses on ICT tools and computational techniques applied for delivering smart mobility services. We identify nine main types of ICT tools, and two computational techniques. Table 3.4 describes each technology and classifies the initiatives that rely on them.

Table 3.4: Technology

Category	Description	Examples
	ICT Tools	

Internet Access	It refers to technologies enabling a computer terminal, mobile device and computer network to connect to Internet for accessing online services. Internet access is provided by various wired or wireless technologies offering a wide range of data transfer speed.	S01, S02, S05, S07, S11, S19-S21, S29-S31, S35, S40, S41
Mobile Broadband	It refers to the various generations (2G, 3G, and 4G) of mobile telecommunications technology for mobile devices and services and networks that fulfil a set of standards defined by the International Telecommunication Union (ITU). It contributes to access services on the move.	S01-S05, S07, S11, S15, S16, S18, S19, S21, S29, S35, S37-S41
Wi-Fi Access Points	It refers to a device that enables wireless devices to connect to a wired network, and to detect and geo-locate Wi-Fi pings made from mobile Wi-Fi enabled devices.	S08, S20
Near Field Communication (NFC)	It refers to a wireless connectivity standard to exchange data using magnetic field induction between two devices located at few centimetres from each other.	S36
Closed Circuit Television (CCTV)	It refers to a TV system, which signals are not publicly distributed but are used to monitor a given area, primarily for surveillance and security purposes.	S20, S22, S26, S27, S32, S35, S42
Global Positioning System (GPS)	It refers to a satellite navigation system enabling to locate an object in longitude, latitude and altitude with high precision.	S01-S04, S05, S07, S10, S11, S16, S18-S20, S39-S41
Radio Frequency Identification (RFID)	It refers to a technology that enables data transfer through wireless use of electromagnetic fields. RFID technology relies on identifying and tracking tags attached to objects.	S08, S10, S19, S24, S28, S33
Smart Sensors	It refers to a device able to sense and convert real-world data into a digital data stream and transmit it wirelessly.	S06, S24, S27, S33, S39
Inductive-Loop Traffic Detector	A kind of smart sensor in the pavement. It is an electrical conducting loop to detect vehicles arriving or passing through.	S22
	Computational Techniques	
Simulation Algorithms	It refers to computational algorithms able to produce models that imitate a current or probable system and its progression.	S09, S33
Video Recognition	It refers to computational techniques enabling to detect, locate and recognize objects and events in a video feed.	S22, S32

3.3.5 Delivery Channels

We define *channel* as the mean used by an administration to interact with and to deliver smart mobility services to its stakeholders. We identify five types of delivery channels. Table 3.5 describes each channel and classifies the services that rely on them for their delivery.

Table 3.5: Delivery channels

Category	Description	Examples
Dynamic Message Sign	It refers to an electronic message board used to provide up-to-the minute information to drivers and the public.	S25, S29, S34
Mobile Device and Applications	It refers to a handheld computing device with capacity for running application software, e.g., smart mobile phone or tablet.	S01-S05, S07, S10, S11, S15, S16, S18, S19, S37-S40
Smart Card	It refers to a plastic card embedded with a microchip and/or NFC facilitating data storage and exchange.	S36
Short Message Service (SMS)	It refers to a text message sent through a mobile device, with a maximum of 160 characters for Latin alphabets, or 70 for Chinese or Arabic alphabets.	S12, S17
Website	It refers to a connected set of Internet pages facilitating access to online resources.	S01, S02, S05, S11, S19, S21, S29-S31, S35, S38, S40, S41

3.3.6 Benefits and Beneficiaries

We define *benefit* as a positive outcome obtained from delivering smart mobility services. We identify direct and indirect benefits delivered to different stakeholders and we classify them according to the smart city dimensions (Section 2.3). In addition, we relate each benefit with public values identified in the literature [92]. Table 3.6 describes the benefits and classifies the services that deliver them.

Table 3.6: Benefits

Category	Description	Public Values	Examples
Smart Economy			

Generating new sources of incomes	It refers to enabling new sources of earnings, e.g., from sharing owned resources.	Citizen's self-development	S01, S05
Generating personal savings	It refers to reducing personal expenditures dedicated to mobility.	Productivity; Effectiveness	S01, S02, S03, S15, S18, S19, S29, S30, S36, S39, S40
Facilitating a sharing economy	It refers to the use of ICT to support the sale or rent of goods and services via online markets. It can be peer-to-peer, business to consumer, etc.	Cooperativeness	S01, S05, S18, S40, S41
	Smart Governance		
Resolving conflicts	It refers to providing evidences to authorities to react faster to resolve conflicts generated by traffic or people.	Effectiveness; Common good; Productivity; Rule of law; Protection of rights of the individual	S20, S26, S27, S32, S42
Detecting illegal behaviour	It refers to facilitating the detection of illegal behaviour committed by drivers.	Rule of law; Effectiveness; Common good	S27
	Smart Mobility		
Facilitating journeys	It refers to enabling journey planning providing alternatives scenarios for moving in the city using different types of transport.	Common good; Public interest; Productivity; Effectiveness	S07, S10, S11, S16, S20, S21, S38
Reducing commuting time	It refers to providing alternatives routes and guidance for reducing the amount of time required to move from one place to another, considering one or more types of transport.	Common good; Public interest; Productivity; Effectiveness; Timeliness	S01, S02, S03, S04, S08, S09, S10, S14, S19, S23, S24, S28,S29, S31, S33-S35, S39, S42
Contributing to reducing traffic congestions	It refers to improving traffic flow.	Common good; Public interest; Effectiveness; Sustainability	S02, S03, S08, S09, S10, S23, S28, S33, S42
Facilitating seamless payment	It refers to facilitating seamless and on-the-move payment of smart mobility services.	Productivity; Effectiveness	S01, S07, S12, S19, S20, S30, S36

Smart Environment			
Reducing CO2 emissions	It refers to contributing to reducing CO2 emissions.	Sustainability; Common good; Public interest; Ethical consciousness	S01, S02, S03, S08, S09, S10, S18, S19, S23, S28, S29, S33, S39, S40, S42
Contributing to becoming a paperless society	It refers to digitizing service delivery, avoiding the use of paper-based forms and interactions.	Sustainability; Common good; Public interest; Ethical consciousness	S01, S07, S12, S15, S19, S20, S30, S36
Using environmentally-friendly transport media	It refers to encouraging the use of means of transport which are classified as environmentally-friendly, like car-pooling, public transport, low emission vehicles, vehicles using environmentally-friendly fuels.	Sustainability; Common good; Public interest; Ethical consciousness	S06, S07, S10, S11, S14, S16, S20, S21, S24, S33, S38
Smart Living			
Improving safety	It refers to improving safety conditions for city dwellers.	Rule of law; Protection of rights of the individual	S18, S20, S25, S32, S34, S35, S37
Improving quality of life	It refers to improving the quality of life of city dwellers in terms of saving time for moving in the city, increasing comfort, simplifying tasks and use of services, etc.	Protection of rights of the individual; Human dignity; Common good	S02-S04, S06, S07, S08, S11-S17, S19-S21, S24, S29, S30, S33, S36, S38, S39
Reducing isolation	It refers to providing access to alternative and more economic transport services in places where public transport is not available, such as access to share rides to move to hospitals, work, etc.	Protection of minorities; Protection of rights of the individual	S05, S18, S40, S41
Developing social values	It refers to the development of social values, like sharing and trust.	Moral standards; Ethical consciousness; Common good	S01, S05, S18, S40
Smart People			

Developing e-skills	It refers to enabling the development of digital skills by using digital services. The substantial benefits provided by digital services could encourage digital illiterates to learn how to use them, while they continue to increase the knowledge of those already digital literate.	Citizens self-development	S01-S05, S07-S12, S15-S24, S26, S29, S30-S33, S35, S37-S42

Associated with benefits, we can identify beneficiaries. A *beneficiary* refers to the actor who receives a benefit from a smart mobility service. From the types of users identified in Section 3.3.3, we identify transport authority, driver, passenger, and resource owner, as beneficiaries. We also identify the society, as beneficiary, representing all kind of social actors. Table 3.7 illustrates this relationship. In addition, due to the faceted structure, we can easily relate information in multiple ways. Thus, we can relate types of services with benefits delivered; as Figure 3.3 shows the type of service contributing to delivering each benefit.

Table 3.7: Benefits and beneficiaries

ID	**Benefits**	**TA**	**D**	**P**	**RO**	**S**
B1	Generating new sources of income		X		X	
B2	Generating personal savings		X	X	X	
B3	Facilitating a sharing economy		X	X	X	
B4	Resolving conflicts		X	X		
B5	Detecting illegal behaviour	X	X			X
B6	Facilitating journeys			X		
B7	Reducing commuting time		X	X		
B8	Contributing to reducing traffic congestions		X	X		
B9	Facilitating seamless payment		X	X		
B10	Reducing CO2 emissions					X
B11	Contributing to becoming paperless society					X
B12	Using environmentally-friendly transport media					X
B13	Improving safety		X	X		X
B14	Improving quality of life		X	X		
B15	Reducing isolation			X		
B16	Developing social values		X	X	X	X
B17	Developing e-skills	X	X	X	X	X

TA = Transport Authority; D = Drivers; P = Passengers; RO = Resource Owners; S = Society

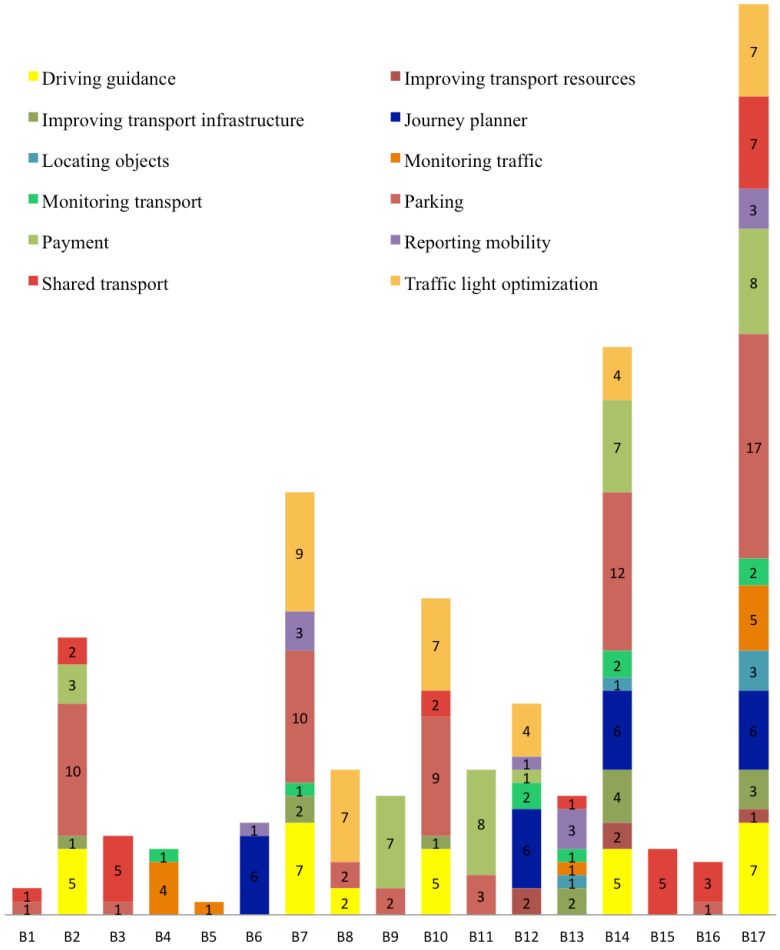


Figure 3.3: Type of service contributing to each identified benefit

3.3.7 Common Functionality

By *common functionality* we refer to a given feature that is present in more than one service, producing different results depending only on different values of certain parameters. Figure 3.4 illustrates this concept, showing the functionality identified for two services, S01) parking sharing (top left) and S05) car sharing (bottom left). In both cases, users request a resource, a parking place or a car, which is near a given location. The object of the request could be potentially any kind of transport-related resource, such as a bus stop, train station or a bicycle. However, the process for handling the request is the same: searching an object in a database that satisfies a given relation, e.g., "*is closed to*" with a given parameter, e.g., a *location*. This is an

example of common functionality used by more than one service. We call such function "*request nearby resource*" and the parameters include *type of resource* and *location*. The common functionality and expected parameters identified for both services is shown in Figure 3.4 (right).

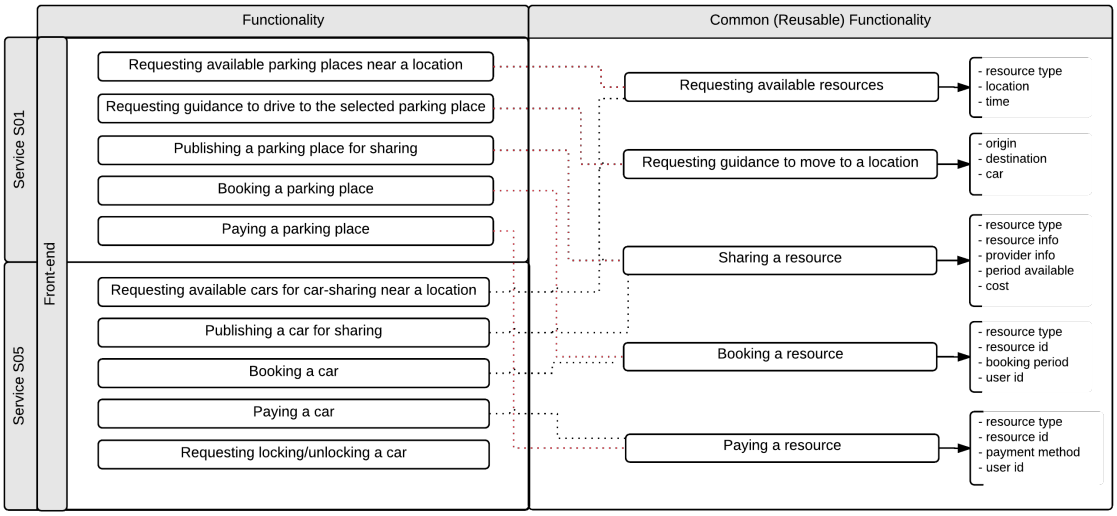


Figure 3.4: Example of common functionality for two smart mobility services.

A summary of the common functionality identified in smart mobility services together with the services that use such functions is presented in Table 3.8.

Table 3.8: Common functionality

Category	**Description**	**Examples**
Requesting near resources	It enables requesting and obtaining from a database all the resources of a given type that are located near a given location.	S07, S16, S21
Requesting available nearby resources	It enables requesting and obtaining from a database all the resources of a given type that are located near a given location and are available at a given timestamp.	S01, S02, S05, S07, S16, S17, S19, S21, S29, S37, S41

Requesting guidance to move to a location	It enables requesting and obtaining concrete instructions to move from a starting point to a destination point by using one or more means of transportation specified. The guidance can be provided with different levels of precision based on different factors, including distance and real time traffic information, among others.	S01-S04, S07, S11, S16, S19, S21, S38, S39
Calculating arrival time	It calculates an estimated arrival time to reach a location from a starting point by using one or more types of transport. The estimated time can have various levels of precision based on different factors, such as distance, real time traffic information, public transport timetables, and geo-location information of a vehicle.	S02, S03, S07, S11, S16, S18, S20, S21, S38
Calculating cost of a service or resource	It calculates the cost of a service or the use of a resource, such as: a journey, a parking place, etc.	S07, S38
Sharing a resource	It enables specifying data about a transport-related resource that is offered to others.	S01, S05, S18, S19, S40, S41
Booking a resource	It enables booking a transport-related resource for a given daytime.	S01, S02, S05, S07, S19, S41
Paying a service	It enables a payment transaction made by the user of a service or a resource to the service provider or resource owner.	S01, S02, S05, S07, S12, S15, S19, S20, S30, S36
Locking and unlocking access to a resource	It enables the locking and unlocking of resources through non-traditional channels (non-manual) such as mobile devices, RFID, etc.	S05, S07, S19
Requesting mobility records	It enables requesting and obtaining records of mobility services used or paid during a given period.	S07, S15, S36
Locating a resource	It enables finding and showing the current location of an object in a map.	S01, S02, S05, S17, S21, S29, S35, S37

3.4 Validation

The taxonomy presented was constructed based on a detailed analysis of the state of research and practice of smart mobility services, following the methodology defined in Section 3.2 and the validation approach described in Section 3.2.4. Below we discuss further decisions underpinning the taxonomy.

The values identified for type of service have been recognized and studied in various

Table 3.9: Type of services – References from literature

Type of Service	**References**
Driving guidance	[106, 155, 156]
Improving transport resources	[21, 67, 152]
Improving transport infrastructure	[29, 68, 121]
Journey planners	[3, 99, 119]
Locating objects	[100]
Monitoring traffic	[30, 100, 135]
Monitoring transport	[77, 102, 153]
Parking	[14, 117, 143]
Payment	[27, 79, 123]
Reporting mobility	[124, 137, 151]
Sharing transport	[69, 129, 132]
Traffic light optimization	[71, 96, 131]

related work, as shown in Table 3.9. The *level of maturity* was based on well-defined existing classifications, as discussed in Section 3.3.2. The values identified for *type of users*, *technologies* and *delivery channels* were identified directly from the description of the surveyed services. No other values were added. The *benefits* were extracted from the description of the services. The criteria for classifying benefits was discussed among the authors, and the *beneficiaries* derived from the service descriptions. While some *common functionality* was extracted from the service descriptions, given the lack of detailed documentation, some other functionality was identified relying on related work [60, 63].

Finally, the focus group meetings enabled to validate the content of the taxonomy. In addition, many valuable ideas to improve the contributions of the taxonomy were recorded and will be considered for future work, including: 1) digitalization of the taxonomy and creation of an online platform where experts can contribute to the evolution of the taxonomy and documentation of case studies; 2) extension of the taxonomy with quantitative benefits, actions taken by the authorities to promote the use of the services, plans for training potential users, dissemination efforts, and information regarding service implementation, such as budget, schedules, staff, etc.; 3) identification of mobility services for people with disabilities, types of service providers, and services using social networks as delivery channel; and 4) categorization of users in terms of daily users such as commuters and students, and occasional users such as tourists.

3.5 Maintenance

The faceted structure provides flexibility to quickly extend the taxonomy horizontally and vertically. New values can be added to each dimension as new initiatives are studied, and new dimensions can be explored as suggested in Section 3.4. Researchers in areas such as e-government, smart cities, and smart mobility are apt to maintain all dimensions except the common functionality dimension, which together with the technology and delivery channels dimensions, would be better maintained by researchers in software engineering and informatics. Below, we briefly discuss the maintainability of the existing dimensions.

Type of services: it could be extended both horizontally and vertically. A hierarchical structure could be suitable for further extending each category, e.g., parking services could be refined as parking sharing and parking search. As new trends and technology emerge, new types of mobility services can be explored, e.g., vehicle-to-vehicle communication services. *Level of maturity*: extensions will depend on innovative uses of ICT to deliver services, and can be extended as proposed in [26]. *Types of users*: it could be extended vertically and horizontally, following either a faceted or hierarchical structure. For example, a faceted structure will enable to distinguished passengers by periodicity (daily commuters, occasional passengers, and tourists) and by whether they possess some disability. A hierarchical structure will be more intuitive to separate drivers into motor-vehicle drivers and bicycle drivers. *Technology*: both sub-dimensions of technology could be extended vertically and horizontally using a hierarchical structure. For example, wireless, cabled and fiber could further refine the category Internet access; and further research following a conceptual to empirical approach could be conducted to identify types of smart sensors, simulation algorithms, and video recognition techniques. *Delivery Channel*: it could be extended horizontally as new delivery channels are identified, such as social media. *Benefits*: it could be extended horizontally and vertically following a hierarchical approach, e.g., detecting illegal behaviour and resolving conflicts could be further refined by type of behaviour and conflict. Further benefits for each smart city dimension can be identified by further analysing smart mobility services. *Beneficiaries*: it can be adapted as new types of users are identified. *Common functionality*: it could be extended horizontally as smart mobility services are further analysed to identify similarities and variability.

3.6 Challenges and Lessons Learnt

One main challenge faced along the taxonomy development process was the difficulty in collecting relevant data about smart mobility services due to the lack of standardiz-

ation and meaningful information provided, as well as a lack of comprehensive lists of initiatives in official government websites. In most cases, we conducted several searches involving different sources to have a clear picture about a given service. This makes it difficult for citizens, researchers and other governments to learn about initiatives and good practices implemented. In addition, it could affect the success of the services being delivered, since citizens may not be aware of their existence. In some cases, governments release reports about the initiatives, providing some organizational and strategic insights. Although these documents are good for other governments, usually they are unfriendly for citizens. Thus, some standardization and trade-off are needed. Citizens would benefit from information about the available services, how they can use them, and how such usage benefits them. Government officials, researchers, and other stakeholders would benefit from information regarding planning, technical details and lessons learnt from smart mobility initiatives successfully or unsuccessfully implemented. Moreover, access to such information could serve open government requirements, now being implemented at subnational level by the Open Government Partnership (OGP)[5]. In particular, the objectives of the OGP initiative include to "discover and promote new and innovative open government techniques and practices emerging at the subnational level around the world; and create practical opportunities for subnational governments to learn from each other, share experiences, and build upon the open government work of their counterparts".

During the data collection process, it was also difficult to find relevant information about the usage of the services. Only when services were delivered through a mobile application, some insights on the number of downloads, users comments, and application rate was available from the application store. In many cases, the number of downloads was not representative with respect to city inhabitants, the rating was average, and there were significant number of complaints regarding the performance of the applications and the unattractiveness of user interfaces. Some recommendations to overcome this, is to have a dedicated strategy to ensure the efficient delivery and usage of smart mobility services. The strategy should strongly rely on citizen engagement, including, as a first step, conducting communication campaigns for promoting the services, informing citizens about their availability and benefits. Other steps should include initiatives for listening to citizens' feedback, facilitating user experimentation as part of innovation labs, documenting the level of user satisfaction, modifying services based on appropriate feedback, maintaining users informed about how their feedback is being used, ensuring the correct maintainability of the software, and collecting data about the actual usage of the services. These efforts should increase trust in the service and in the service provider, and enhance the level of acceptance and usage of the service.

[5] http://www.opengovpartnership.org/how-it-works/subnational-government-pilot-program

In many cases, information about the stakeholders involved in the development and delivery of the services and about the level of government engagement was missing. Many of the services were provided by the public, as mechanisms for promoting a collaborative or sharing economy, e.g., most parking services were developed without government involvement. In all such cases, it is recommended that governments revise their roles as regulators and as providers of the needed platforms for promoting the development and delivery of such type of services by entrepreneurs and representatives of the private sector [89]. Finally, since the data was collected, we noticed that some government websites have removed information about the services. The reasons could rely on several arguments, such as failure of the initiative, and closure after achieving its goals. To facilitate knowledge sharing, it would be helpful to maintain a knowledge base, accessible to the public, containing data and lessons learnt about conducted initiatives, whether they are active or not. This is important for other governments and researchers, but also for citizens, as accountability mechanism.

3.7 Related Work

Some taxonomies exist in the literature covering smart city concepts, for instance: a taxonomy of application domains for smart cities, including transport and mobility domain that is further classified into city logistics, info-mobility, and people mobility [118]; a taxonomy to classify smart city projects comprising the description of the project, the business model, and the purpose [126]; and a taxonomy of technologies for smart cities [154].

Recently, a taxonomy of smart mobility has been proposed by Benevolo et al. [24]. Such a taxonomy intersects with the one proposed in this article in the types of services and benefits dimensions, and in some values identified for such dimensions. We believe that the taxonomy proposed here provides a wider view of smart mobility services by considering other dimensions related to service development. In addition, main differences between both taxonomies include: we recognize benefits for all dimensions of a smart city, while the other authors focus on benefits for smart mobility; we focus in services highly relying on ICT, while Benevolo et al. consider a broader range of solutions, such as vehicles depending on sustainable fuels and policy actions; and we develop the taxonomy based on the state of art assessed from literature review and 42 initiatives from smart cities around the world, while Benevolo et al. focus on economic papers regarding policies and technologies for urban and smart mobility, especially in Europe.

3.8 Conclusions

The contribution of the taxonomy is twofold. First, it presents a broad mapping of mobility services that can be deployed in the context of smart city initiatives. Second, it introduces a taxonomy defining and classifying relevant concepts for policy makers and software engineers.

On the one hand, policy makers can benefit from the taxonomy when defining smart mobility strategies, since it enables the identification of stakeholders to whom services need to be defined, exemplifies the different type of services to be delivered, and the corresponding benefits and beneficiaries, facilitating the justification of business cases for each initiative. In addition, entrepreneurs can benefit from the taxonomy to study possible market opportunities to innovate in the provision of smart mobility services. On the other hand, software engineers can benefit from the identification of common functionality that can be used to develop reusable components for smart mobility services. We discuss in more detail applications of the taxonomy and some usage scenarios in Section 8.2 when we discuss the planning component of the virtual factory.

As a limitation, the analysis was based on secondary data. Such data was gathered from government websites and reporting documents as well as from scientific publications.

Chapter 4

Technical Background

This chapter gathers the necessary technical background for the thesis and comprises three main concepts.

First, in line with the goal of rapid developing government services, we discuss the notion of Software Product Lines (SPLs), an approach to model families of software systems that takes advantage of shared functionality to rapidly develop them. There are two main approaches to model variability in SPLs, here referred as *fine-grain* and *coarse-grain*. However, we argue that a third *compositional* approach can be recognized from the literature. This approach seems to gather the best from the fine-grain and coarse-grain approach, thus, we proposed a compositional approach to model SPLs in Chapter 5.

Second, we present a promising existing fine-grain formalism to model SPLs, called Featured Timed Automata, in which we based our proposed formalism to model SPLs in a compositional way.

Finally, we present $\mathcal{R}$eo, and exogenous coordination language to orchestrate distributed components. One of the main advantages of $\mathcal{R}$eo, is that it allows to yield loosely-coupled systems where components comprising domain functionality are separated from the protocols that orchestrate them. Thus, it seems interesting to incorporate the notion of exogenous coordination with SPLs to decoupled coordination from domain functionality and improve the reusability of both, domain assets and coordination protocols. To to this, it is necessary to model $\mathcal{R}$eo protocols as families of protocols, as we do in Chapter 7.

Chapter Organization. Section 4.1 presents the definition of software product lines (SPLs), discusses their benefits, some common approaches to formally model SPLs, and their relevance to support Digital Government. Section 4.2 discusses Featured Timed Automata, a formalism to model SPLs that supports real-time requirements.

Finally, Section 4.3 presents Reo, an exogenous coordination language, and discuss the advantages of incorporating the notion of exogenous coordination in SPLs.

4.1 Software Product Lines

Software development has gone through various stages, from custom made, to mass production, to mass customization [7, 45, 127]. In the beginning, software was *custom made* for a specific hardware, purpose, or client. As the production of hardware increased and software became more complex, software producers agreed on standard platforms as a way to develop standard software and off-the-shelf components that could be installed by many, bringing *mass production* to the software industry. However, as the proliferation of devices and software increased, so did customer demand to receive better and customized products, something that mass production could not deal with. This gave rise to the adoption of *mass customization* – large scale production of goods tailored to individual customers' needs [51].

Software Product Lines (SPLs) emerged as a way to bring mass customization to the software industry [7], similar to how production lines brought mass customization to other markets, such as car manufacturing.

4.1.1 Definition (Software Product Line (SPL) [45]). A *software product line* is a set of software-intensive systems sharing a common, managed set of *features* that satisfy the specific needs of a particular market segment or mission and that are developed from a set of core assets in a prescribed way. ■

An SPL enables the development of a family of software systems by taking advantage of the commonalities and variabilities of the members of the family. These commonalities and variations are referred as features. There are many definitions in the literature of what a feature is, here we use the following one.

4.1.2 Definition (Feature [7]). A *feature* is a characteristic or behaviour of the system visible to the user. It captures both requirements of the end users as well as implementation concepts. ■

By selecting a desired set of features, one can derived a concrete *product* from the SPL. The set of all valid *feature selections* determines the *scope* of the software product line, i.e. the set of products that can be *derived* from the SPL. These valid combinations are specified through variability models, called *feature models*. We discuss feature models and their representation in Section 4.1.1.

The set of core assets includes the typical artefacts in software development: requirements, domain models, test cases, software architecture, and components, among

others, all of which are designed to accommodate variability and to be reused. Unlike with merely reusability of existing components, in SPL reusability and variation points are carefully *planned, enabled* and *enforced* [45]. Through out this work we focus only on models.

In addition to mass customization, following an SPL approach brings other important benefits, such as: *reducing costs and time to market* – although the upfront investment is larger than developing a single product, it pays off in the long term, since there is no need to develop customized solutions from scratch. Developing a product reduces to selecting the desired set of features. Even if it may be necessary to tweak a final product to accommodate new functionality, the efforts and costs are lower; and *improving quality* – the reusability and testing of assets in many products leads to more stable and reliable products than if they were built from scratch; among others.

The increase in benefits achieved through an SPL entails new risks and significant development efforts during start-up, as well as ongoing costs to maintain the assets. An example of a typical risk that can arise has to do with the definition of the SPL scope. In fact, in [7] the authors discuss that a well-defined and well-scoped domain is a key success factor for SPL development. If the scope of the SPL is too wide, it will lead to too much variability, making it difficult to accommodate variability, and resulting in going back to a product by product development approach. If the scope is to small, the core assets might not be generic enough, making it difficult to accommodate future growth, stagnating the development, and leading to no return on investment.

To avoid these issues, understanding correctly the domain of application is of utmost importance. Thus, the software engineering process of an SPL is divided into two main processes, each of which incorporates two main activities: *domain engineering* – deals with domain analysis and domain implementation; and *application engineering* – deals with requirements analysis and product derivation. This is schematized in Figure 4.1 taken from [7].

Essentially, the activities involved in each of the main tasks are described as follows. 1) *Domain analysis* – identifies the scope of the domain and the features that are relevant and should be implemented as reusable assets. This is typically documented in a feature model (see Section 4.1.1). 2) *Requirements analysis* – studies the requirements of a given client. This can result in a simple feature selection, or it might result in new requirements, leading to expand the domain analysis and possible adaptation of the feature model. 3) *Domain implementation* – develops the reusable assets that correspond to the features identified in domain analysis. 4) *Product derivation* – is the production step of application engineering, where reusable artifacts are combined according to the results of requirement analysis. Depending on the implementation approach, this process can be more or less automated, possibly, involving several development and customization tasks.

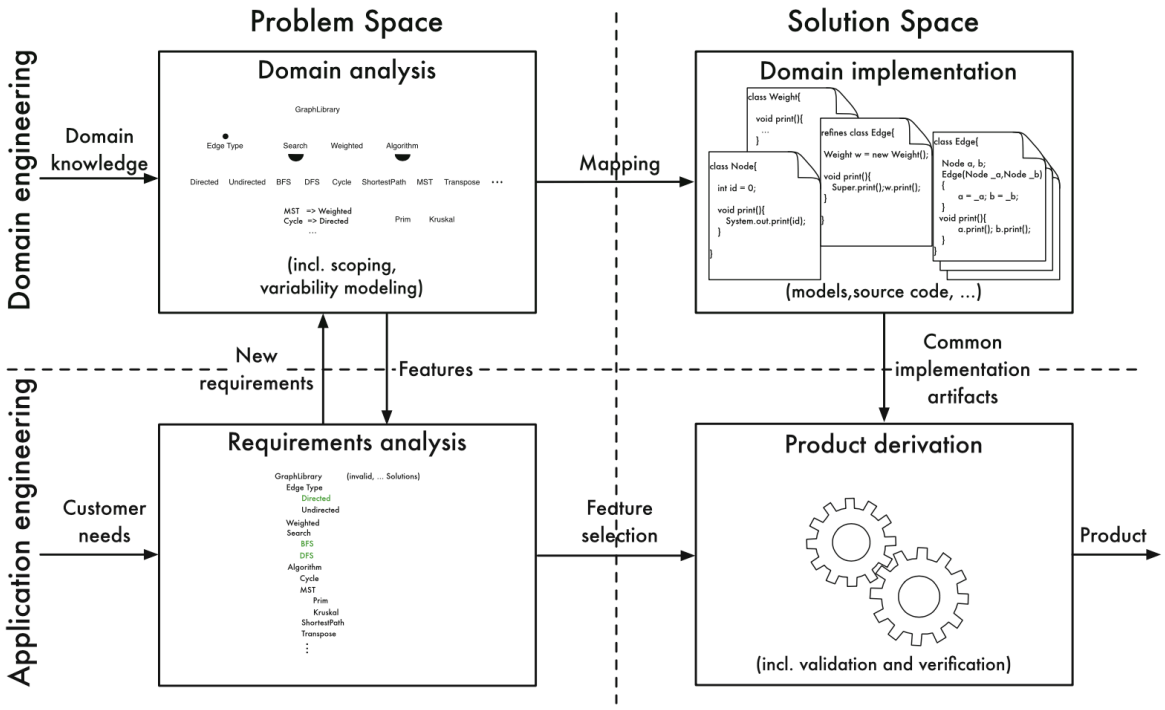


Figure 4.1: The engineering process for software product lines [7].

4.1.1 Variability

Variability is identified and documented during domain analysis, and it is used during the application engineering process to derive concrete products of the family. The variability is typically defined in terms of common (present in every product) and optional features, usually through variability models, called *feature models*. A feature model expresses the valid combination of features by defining dependencies between them, where each combination is a product supported by the family.

There are various approaches to express feature models, both in textual description and graphical representation. The latter are usually referred as *feature diagrams*. Graphical representations however do not provide a formal semantics. A common way to give formal semantics to a feature diagram is by mapping it directly into a propositional logic [7, 23, 110].

Feature Diagrams

Among feature diagrams, there are many variants as well, including the first definition [94] and many extensions to this notation [75, 95]. In [134] the authors study

various notations for feature diagrams, providing a formal semantics for these and analysing their expressiveness, succinctness and embeddability, concluding that many of the existing extensions do not increase expressiveness and sometimes are ambiguous. Similar, in [34] the authors survey various approaches to model feature diagrams and provide a textual notation with formal semantics.

Here we adopt the notation used in [7]. In this case, a feature diagram is represented as a tree structure. Some notations allow the diagram to be a graph (see [134]). Each node of the tree represents a feature. There are various types of notations that link non-leave nodes with their children, expressing a type of constraint between them.

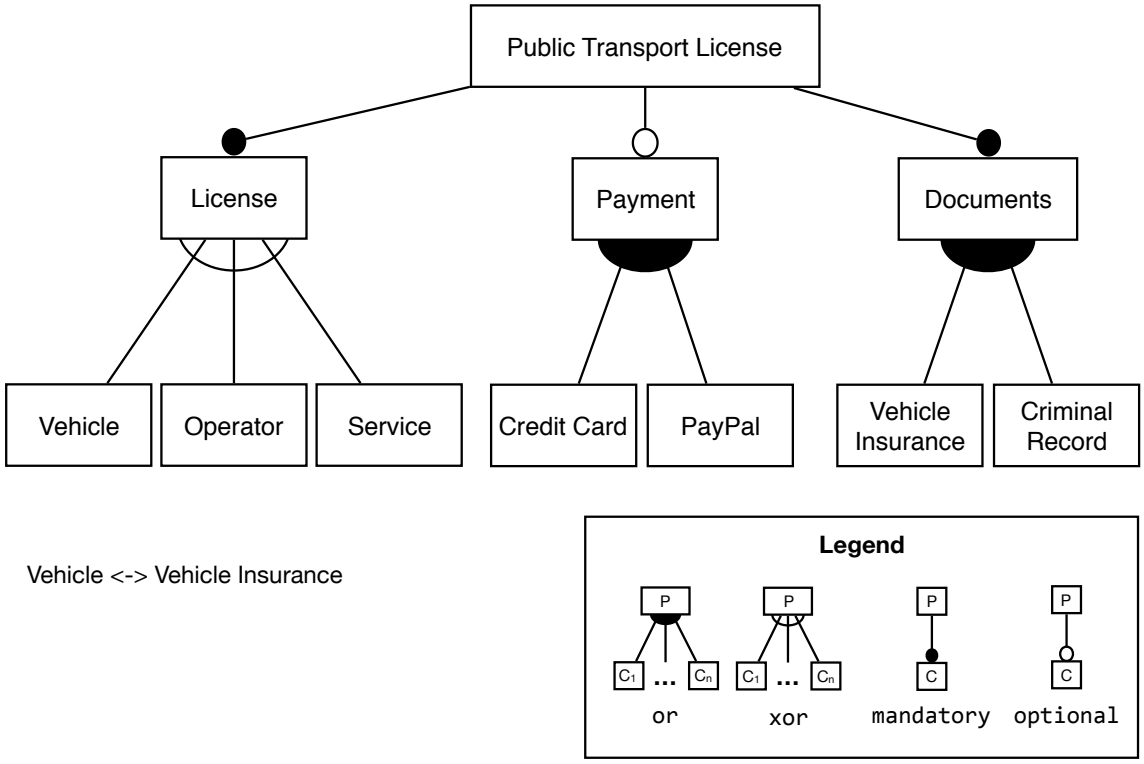


Figure 4.2: An example of a feature diagram for an SPL of public transport licensing services.

Figure 4.2 illustrates a simplified feature diagram extracted from the case study shown in Chapter 8. A root feature is always present in every product, in this case *PublicTransportLicense*. A relation between a feature and its child features can be of

four types as described by the legend: `or` – some out of many can be selected; `xor` – one out of many can be selected; `mandatory` – always present; and `optional` – optionally present. For example, a feature *License* must always be selected, specifying *only one* type of license, *Vehicle*, transport *Operator*, or *Service*. Not all public transport licenses require *Payment* support, and if they do, they could support payments through *CreditCard*, *PayPal*, or both. In addition, a feature can be selected only if its parent feature is selected. Because not all constraints can be expressed in a tree structure, additional *cross-tree* constraints can be added in textual form using propositional logic. In the example, support for *VehicleInsurance* documentation can be selected, if and only if, the selected license to be derived is a license for *Vehicles* to transport passengers.

As mentioned, there are many variants of feature diagrams, some distinguish between concrete and abstract features, where an abstract feature do not have associated assets, others allow multi-features [48], i.e. to select a feature multiple times, each time with different child configurations. A comprehensive analysis of feature diagrams can be found in [134].

Formal feature models

The simplest way to give a formal semantics to feature diagrams is to translate them into propositional logic where each feature is treated as a boolean variable. Examples of this approaches can be seen in [7, 23, 110].

Given a feature diagram over a set of features F, we can obtain a formal feature model in propositional logic applying the following set of rules to each edge of the feature diagram [7]:

$$
\begin{aligned}
\texttt{root}(P) &\equiv P \\
\texttt{mandatory}(P, C) &\equiv P \leftrightarrow C \\
\texttt{optional}(P, C) &\equiv C \rightarrow P \\
\texttt{or}(P, \{C_1, \ldots, C_n\}) &\equiv P \leftrightarrow (C_1 \vee C_2 \vee \cdots \vee C_n) \\
\texttt{xor}(P, \{C_1, \ldots, C_n\}) &\equiv P \leftrightarrow (C_1 \vee \cdots \vee C_n) \wedge \bigwedge_{i \leq j} \neg(C_i \wedge C_j)
\end{aligned}
$$

The resulting future model in propositional logic is the conjunction of formulas obtained by applying such rules. In addition, since cross-tree constraints can be added in textual form to the feature diagram, these are as well conjoined in the resulting model.

For example, the feature diagram shown in Figure 4.2 can be translated into the

following propositional formula:

$$
\begin{aligned}
fm \equiv & \mathit{PublicTransportLicense} \\
& \wedge \mathit{License} \leftrightarrow \mathit{PublicTransportLicense} \\
& \wedge \mathit{Documents} \leftrightarrow \mathit{PublicTransportLicense} \\
& \wedge \mathit{Payment} \rightarrow \mathit{PublicTransportLicense} \\
& \wedge \mathit{License} \leftrightarrow (\mathit{Vehicle} \vee \mathit{Operator} \vee \mathit{Service}) \\
& \wedge \neg(\mathit{Vehicle} \wedge \mathit{Operator}) \wedge \neg(\mathit{Vehicle} \wedge \mathit{Service}) \wedge \neg(\mathit{Operator} \wedge \mathit{Service}) \\
& \wedge \mathit{Payment} \leftrightarrow (\mathit{CreditCard} \vee \mathit{PayPal}) \\
& \wedge \mathit{Documents} \leftrightarrow (\mathit{VehicleInsurance} \vee \mathit{CriminalRecord}) \\
& \wedge \mathit{Vehicle} \leftrightarrow \mathit{VehicleInsurance}
\end{aligned}
$$

Similar to [46] and for simplicity, throughout this work we formally defined a feature as a boolean variable that represents a unit of variability; and defined a feature model as a boolean expression over features, called *feature expressions*. Formally, feature expressions, their satisfaction and semantics are defined as follows.

4.1.3 Definition (Feature Expressions (FE), satisfaction, and semantics). A feature expression φ over a set of features F, written $\varphi \in FE(F)$, is defined by

$$\varphi ::= f \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \top \qquad \text{(feature expression)}$$

where $f \in F$ is a feature. The other logical connectives can be encoded as usual: $\bot = \neg\top$; $\varphi_1 \rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$; and $\varphi_1 \leftrightarrow \varphi_2 = (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$.

Given a feature selection $FS \subseteq F$ over a set of features F, and a feature expression $\varphi \in FE(F)$, FS *satisfies* φ, noted $FS \models \varphi$, if

$$
\begin{array}{ll}
FS \models \top & \textit{always} \\
FS \models f & \Leftrightarrow f \in FS \\
FS \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow FS \models \varphi_1 \textit{ and } FS \models \varphi_2 \\
FS \models \varphi_1 \vee \varphi_2 & \Leftrightarrow FS \models \varphi_1 \textit{ or } FS \models \varphi_2 \\
FS \models \neg\varphi & \Leftrightarrow FS \not\models \varphi
\end{array}
\qquad \text{(FE satisfaction)}
$$

The semantics of a feature expression φ with respect to a set of features F, denoted $[\![\varphi]\!]^F$, is the set of valid feature selections over F that satisfy φ, formally,

$$[\![\varphi]\!]^F = \{FS \subseteq F \mid FS \models \varphi\} \qquad \text{(FE semantics)}$$

We will use simply $[\![\varphi]\!]$ when F is clear from the context. ■

Thus, the semantics of a feature model is the set of products that can be derived from it, i.e. the valid combinations of features. If we consider the boolean formula obtained from the feature model in Figure 4.2, there are 16 valid combinations of features allowed by this feature model.

We model feature expressions in a constraint fashion, i.e., a feature expression expresses the minimum requirement over a set of features that needs to be satisfy. For example, a feature expression $\varphi = \top$ over the set of features $F = \{f_1, f_2\}$, expresses that any combination of features f_1 and f_2 is possible, in fact $[\![\varphi]\!]^F = \mathbf{2}^F$, while the feature expression $\varphi' = f_1 \rightarrow f_2$, over F, expresses that whenever f_1 is selected, f_2 must be selected, being $[\![\varphi']\!]^F = \{\{\}, \{f_2\}, \{f_1, f_2\}\}$.

4.1.2 Modelling Software Product Lines

One of the main challenges in SPL is to guarantee efficiently that every product of the SPL behaves as expected. The issue is that in practice software product lines can deal with thousands of features and thousands of valid feature combinations. In fact, the number of products that can be derived from an SPL is exponential in the number of features. One approach towards addressing this issue is to have modular and scalable behavioural formalisms to model and verify properties of the SPL.

When modelling SPLs, one of the aspects to consider is how variability is associated to specific assets, in our case models, of the SPL. There are two main approaches recognized in the literature to model and verify SPLs [7, 97] that differ in the level of granularity of the assets to which variability is assigned. For example, in an object-oriented program, variability could be assigned to a class, a method, or a statement, among others. These two approaches are refer as annotative and compositional. However, we argue that the term compositional can be misleading, since in practice, both approaches are annotative in the sense that assets in the SPL are annotated with the features they belong to. Here we refer to them as *fine-grain*, and *coarse-grain* approach, respectively. In practice, both approaches can be combined, as in Kästner et al. [97]. Below we discuss a third approach to model variability, which we argue is truly compositional.

In an fine-grain approach, the behaviour of all the products of the family is represented in a single model. Parts of the model are then annotated with variability specifying in which products that part is present. When selecting a specific product, only the parts of the core that belong to that product are preserved, the rest are pruned or ignored. This has the advantages and disadvantages of any fine-grain approach. On the one hand, it allows greater flexibility to model cross-cutting functionality. In addition, it allows to verify properties of the entire family instead of following a product by product approach. On the other hand, it does not allow to define an SPL in a

modularized way. The core asset can quickly grow in complexity, which hinders the understanding, maintainability, and evolution of the SPL. Examples of fine-grain behavioural formalisms found in the literature include the use of transition systems and finite automata [35,47], where transitions are annotated with a feature expression, similar to how feature expressions are defined here. In both cases, a single model is built containing the behaviour of the entire family. Model checking techniques are later used to verify properties over the entire model.

In a coarse-grain approach, the modelling and reasoning of features is done in isolation. The SPL is design in a modularized way, where functionality is divided into composable units, where each unit typically corresponds to a feature. Each feature specifies how it alters the core model. When selecting a specific product, only components that correspond to a selected feature are composed into the final product. It can be verify if a given combination of features preserves the properties of the individual ones, but requires to check over every possible combination of features. It has been proposed [8] to model features in a compositional manner but to model check them following an fine-grain approach. The advantage of the coarse-grain approach is that variability and functionality are encapsulated in composable units, simplifying the design, and addressing the main issues of the fine-grain approach. Of course, the main disadvantage is that it is not always possible to decompose variability into well-defined components, as in the case of cross-cutting functionality. Typical examples of this approach can be found at the level of implementation [22, 97], where features can extend a class by redefining methods or adding new ones.

From the literature, we would like to recognize a third approach, a truly *compositional* approach. In this approach, an SPL can be designed following a component-based approach in which functionality is divided into simpler components, each of which can be annotated with variability. In addition, each component possess its own feature model. The SPL can then be built by composing the components, including their variability models. An example of a behavioural formalism following this approach includes an extension to Petri Nets, where each net has a feature model, and arcs in the net can be annotated with features indicating in which products the arcs are present [115]. Various nets can be combined through composition, where feature models from separately engineered models of individual features are composed as well.

In the coarse-grain approach, only components that belong to selected features are combined in the final product. In this sense, this approach composes products. In the truly compositional one, the entire SPL is composed out of smaller variable components. In this sense, this approach composes SPLs, or families of products. This provides some advantages over only fine-grain or coarse-grain approaches. First, it gathers the best of both approaches: modularization while having the ability to assign variability at a finer level. Second, it allows to model variable components

with independent variability models, composed them together and reason about the composed variability model, what products can be derived, and dependencies between features, among others. Thus, in this work we focus on the compositional approach and propose accordingly a compositional formalism to model SPLs in Chapter 5.

4.1.3 SPLs and Digital Government

Only a few studies can be found in the literature regarding SPL support for the development of digital government applications. In [107], the authors propose an SPL for generating front-end environments for an e-government context management system. In [125], the authors propose a method to generate personalized government documents using SPL. The approach takes advantage of the high level reuse of government documents. More recently, SPLs have been used to support the development of personalized applications in smart cities [105].

Although not yet widely used in government, the concept of SPL offers unique opportunities for the rapid development and deployment of certain services provided by local governments. For example, following our case study, in most public administration systems, local governments are responsible for issuing licenses for public transport services. While such services comprise a number of common functionality – e.g. submission and approval of the public transport route, submission and approval of the vehicles certificates of property, submission and approval of the drivers' driving licenses, etc.; they also differ in a number of features, mostly due to specific regulations of each local government. Therefore, the idea of applying SPL for generating a family of certain type of public services is appealing and relevant to the public sector.

However, in addition to the concept, there is a need to provide appropriate tools to facilitate the generation of reliable software systems supporting the delivery of public services based on the SPL concept. In this sense, the use of formal methods and SPLs can help to overcome this issue and, to some extent, the challenges described in Section 2.2. On the one hand, formal methods help to model and verify that services conform with the required laws and regulations at an early stage. On the other hand, software product lines can help to rapidly develop families of services, reducing costs and development efforts, as well as facilitating service integration. Thus, it offers unique opportunities for the rapid development and deployment of certain services provided by local governments.

Thus, integrating both approaches seems the way to go. Various formalisms and approaches exist to formally model and verify SPLs as mentioned in Section 4.1.2. However, they are mainly fine-grain or coarse-grain approaches. Following a compositional approach to formally model SPLs seems more interesting due to its flexibility and potential improvement over the other approaches.

In addition, based on our case study we recognize that time constraints, although not necessarily real-time constraints, are a recurrent requirement in government services. Thus, we focus in a formalism capable of capturing this requirements. From the literature, Featured Timed Automata is a very interesting formalism with efficient algorithms to verify properties over the models. However, it is a fine-grain approach, and as such it does not allow to model SPLs in a compositional way. Thus, the compositional formalism proposed in Chapter 5 is based on this formalism. We provide the necessary background on Featured Timed Automata in the following section.

4.2 Featured Timed Automata

Featured Timed Automata (FTA), introduced by Cordy *et al.* [47], is an extension to Timed Automata (TA) [4] to model and verify families of real-time systems parametrized by a variability model. This parametrization enables to model various systems in a single model, simplifying the design and improving efficiency during model checking.

Essentially, FTA extends TA by incorporating a set of *features*, which represent units of variability, and a *feature model*, which determines the set of valid feature combinations. Each valid combination determines a concrete product in the family, i.e., a timed system modelled as a timed automata.

We first introduce some preliminary notions of Timed Automata in Section 4.2.1. Then, in Section 4.2.2 we present the formal definition of a featured timed automaton and its semantics based on Cordy *et al.*.

4.2.1 Timed Systems

Featured Timed Automata builds on top of Timed Automata, introduced by Alur *et al.* in [4] as an abstract model for timed systems. A timed automaton is a finite automaton equipped with a finite set of real-valued variables representing a set of *clocks*, and logic guards over clocks, called *clock constraints*, to model time constraints about the behavior of the system.

A clock c is a logical entity that models continue and dense-time. It can only be inspected or reset to zero, and represents the time elapsed since its last reset. When a timed automaton evolves over time, all clocks are incremented simultaneously. Initially, all clocks are reset to zero.

Clock constraints are logic conditions over the value of a clock. They are used as logic guards to represent time constraints over locations, restricting the amount of time the system can spend in the location; and over transitions, restricting when the system can move to another location by taking an action on a transition. Specifically, clock

constraints over locations are called *location invariants*.

Intuitively, a TA may remain in a location as long as the current *clock valuation*, i.e., the current value of the clocks, satisfies the invariant of the location; and may leave a location by taking an outgoing transition, only while the current clock valuation satisfies the clock constraint of the transition. Notice that transitions are instantaneous steps, i.e., taking a transition does not increment the value of the clocks. We illustrate this with an example before presenting any formal definition.

4.2.1 Example. Figure 4.3 shows a timed automaton modelling a payment selection controller. There are two locations, ℓ_0 and ℓ_1, a clock c , and three actions *pay*, *creditcard* and *paypal*, representing the ability to pay, and to select the payment method, i.e., credit card or PayPal, respectively. Initially the automaton is in location ℓ_0, indicated by a node with double line [1], and it can evolve either by waiting for time to pass, incrementing the clock c, or by taking the transition to ℓ_1. Taking such transition triggers the reset of the clock c back to 0, evolving to the state ℓ_1. Here it can wait for the time to pass, but for at most 5 time units, determined by the invariant $c \leq 5$ in ℓ_1. In location ℓ_1 it can either evolve by selecting to pay by PayPal or by credit card. However, these transitions have a different guard: a clock constraint $c \geq 1$ that allows the transitions to be taken only when clock c is greater than 1. ◁

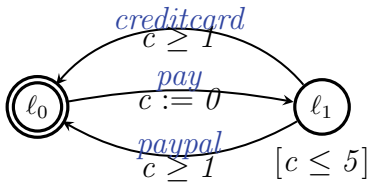


Figure 4.3: An example of a TA modelling a payment selection controller.

We now present the formal definition of clock constraints and timed automata.

4.2.1 Definition (Clock Constraints (CC), valuation, and satisfaction). A *clock constraint* over a set of clocks C, written $g \in CC(C)$ is defined by

$$g ::= c < n \mid c \leq n \mid c = n \mid c > n \mid c \geq n \mid g \wedge g \mid \top \qquad \text{(clock constraint)}$$

where $c \in C$, and $n \in \mathbb{N}$.

[1] This notation is adopted as well by UPPAAL. It can be confusing since in *Büchi Automata* a double line indicates accepting states, which are used to reason about the progress of the system. However, when dealing with *Timed Safety Automata*, as in this case, the notion of progress is given by invariants associated to locations instead of accepting states [25].

A *clock valuation* η for a set of clocks C is a function $\eta\colon C \to \mathbb{R}_{\geq 0}$ that assigns each clock $c \in C$ to its current value ηc. We use $\mathbb{R}^C$ to refer to the set of all clock valuations over a set of clocks C. Let $\eta_0(c) = 0$ for all $c \in C$ be the initial clock valuation that sets to 0 all clocks in C. We use $\eta + d$, $d \in \mathbb{R}_{\geq 0}$, to denote the clock assignment that maps all $c \in C$ to $\eta(c) + d$, and let $[r \mapsto 0]\eta$, $r \subseteq C$, be the clock assignment that maps all clocks in r to 0 and agrees with η for all other clocks in $C \setminus r$.

The *satisfaction* of a clock constraint g by a clock valuation η, written $\eta \models g$, is defined as follows

$$\begin{array}{ll} \eta \models \top & \text{always} \\ \eta \models c \,\square\, n & \text{if } \eta(c) \,\square\, n \\ \eta \models g_1 \wedge g_2 & \text{if } \eta \models g_1 \text{ and } \eta \models g_2 \end{array} \qquad \text{(clock satisfaction)}$$

where $\square \in \{<, \leq, =, >, \geq\}$. ■

4.2.2 Definition (Timed Automata (TA)). A *timed automaton* is a tuple $\mathcal{A} = (L, \ell_0, A, C, T, Inv)$, where:

- L is a finite set of locations,
- ℓ_0 is the initial location,
- A is a finite set of labels representing actions,
- C is a finite set of clocks,
- $T \subseteq L \times CC(C) \times A \times \mathbf{2}^C \times L$ is a finite set of transitions
- $Inv : L \to CC(C)$ is a function assigning invariants to locations.

In a transition $(l, g, a, r, l') \in T$, sometimes written $\ell \xrightarrow{g,a,r}_{\mathcal{A}} \ell'$, l is the origin location, g is the clock constraint or *guard* associated to the transition, a is the action triggering the transition, r is the set of clocks to reset, and l' is the target location. ■

Notation. $\mathbf{2}^A$ denotes the powerset of A. Given any automaton $\mathcal{A}$, we use $L_{\mathcal{A}}, \ell_{0_{\mathcal{A}}}$, $A_{\mathcal{A}}, \ldots$ to refer to the elements of $\mathcal{A}$ when not clear from the context.

The semantic representation of a TA is given by a *Labelled Transition System* or just Transition System (TS) [16]. Figure 4.4 illustrates the infinite transition system corresponding to the TA of the payment controller shown in Figure 4.3. The states of the underlying TS of a TA $\mathcal{A}$ consist of a location of $\mathcal{A}$ and a clock valuation that represents the current value of the clocks in that state. The transitions between states

of the TS can be *delay transitions*, e.g., $\langle \ell_0, c = 0 \rangle \xrightarrow{2} \langle \ell_0, c = 2 \rangle$, or *action transitions*, e.g., $\langle \ell_0, c = 0 \rangle \xrightarrow{pay} \langle \ell_1, c = 0 \rangle$. Delay transitions model the passing of time, i.e., they model when $\mathcal{A}$ does not take any action, staying in the same location but incrementing the value of the clocks. Action transitions model when $\mathcal{A}$ takes an action on an actual transition, moving to a possible new location and without altering the value of the clocks, except for those that are reset to 0 by the transition.

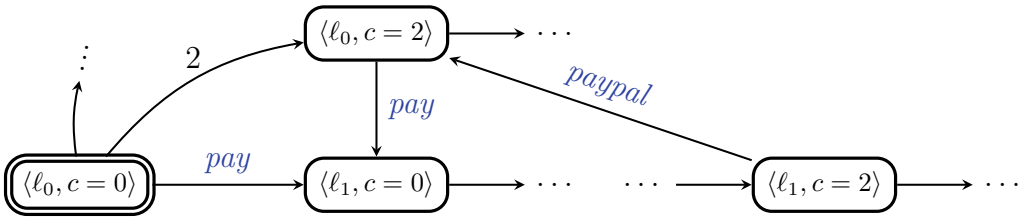


Figure 4.4: An illustration of an infinite TS corresponding to the TA in Figure 4.3.

4.2.3 Definition (Semantics of TA). The *semantics of a timed automaton* $\mathcal{A} = (L, \ell_0, A, C, T, Inv)$, is a transition system $[\![\mathcal{A}]\!] = (St, s_0, A, T')$, where:

- $St = L \times \mathbb{R}^C$ is the set of states, consisting of the current location and clock valuation,
- $s_0 = \langle \ell_0, \eta_0 \rangle$ is the initial state, and
- $T' \subseteq St \times (A \uplus \mathbb{R}_{\geq 0}) \times St$ is the transition relation, defined as follows

$$\langle \ell, \eta \rangle \xrightarrow{d} \langle \ell, \eta + d \rangle \text{ if } \eta \models Inv(\ell) \text{ and } (\eta + d) \models Inv(\ell), \text{ for } d \in \mathbb{R}_{\geq 0} \quad (4.1)$$

$$\langle \ell, \eta \rangle \xrightarrow{a} \langle \ell', \eta' \rangle \text{ if } \exists\, \ell \xrightarrow{g,a,r} \ell' \in T \text{ s.t. } \eta \models g, \; \eta \models Inv(l), \; \eta' = [r \mapsto 0]\eta, \text{ and } \eta' \models Inv(\ell') \quad (4.2)$$

■

Condition (4.1) defines delay transitions. A delay transition is labelled with a positive delay $d \in \mathbb{R}_{\geq 0}$. For a delay transition to exist, the clock valuation of the origin state, η, must satisfy the invariant of the location ℓ; while the updated clock valuation after taking the delay transition, $\eta + d$, must still satisfied the invariant of such location. Condition (4.2) defines action transitions. The transition is labelled with the action of the corresponding transition in $\mathcal{A}$. For an action transition to exists between two

states, the clock valuation of the origin state, η, must satisfy the invariant of the origin location, ℓ, and the guard, g, associated to the transition; while the clock valuation of the target state, defined as η for all clocks, except for clocks in r which are reset to 0, must satisfy the invariant of the target location, ℓ'.

Multiple TA can be composed in parallel to model concurrent systems, resulting in a *network of Timed Automata*. Parallel composition of TA reduces to computing the product of the automata, where the automata in the network may interleave or transition together through the use of shared actions, called *synchronization actions*. In a network there are no external actions, i.e., all actions are synchronized and the network is considered a closed system. The formal definition of a network of TA and its semantics can be found in [25]. In practice, computing the product of a network of automata is computationally expensive, however, it can be computed on-the-fly as is in the case of the real-time model checker UPPAAL.

4.2.2 Families of Timed Systems

Featured Timed Automata is an *fine-grain* approach introduced by Cordy *et al.*, [47] to model real-time SPL. Thus, an FTA encodes in a single model various models corresponding to a family of timed systems, in particular, each model being a concrete timed automaton. Given a FTA, it is possible to model check it against a given property expressed in Timed Computation Tree Logic (TCTL) [16] and determine the set of products that satisfy or violate such a property.

In order to model a family of timed automata in a single model, FTA extends TA with *features*, *feature expressions*, and a *feature model*. As mentioned in Section 4.1.1, features represent units of variability supported by the model, and are modelled as a set of Boolean variables. Feature expressions are Boolean conditions over features, and are associated to transitions to specify the set of concrete products of the SPL (in this case, concrete TA) where a given transition is present. The feature model, here modelled as a feature expression, represents the set of valid feature combinations, i.e., the set of concrete products that can be derived from the model.

We can now present the formal definition of FTA based on Cordy *et al.*. For the sake of simplicity, we have slightly modify the original definition. In particular, we use an unique initial location instead of a set of locations; we use clock constraints instead of *featured clock constraints* (clocks constraints parametrized by features expressions); and we do not associate *atomic propositions* to locations. In the last case, this is because FTA uses a *state-based* approach to verify properties of a family, where the observable behaviour of an FTA is given by a set of properties, usually called atomic propositions, associated to the current state of the system. However, we use an *action-based* approach, since we focus on modelling interactive systems where the observable

behaviour is rather given by the actions available through outgoing transitions from the current state.

4.2.4 Definition (Featured Timed Automata (FTA) [47]). A *featured timed automaton* is a tuple $\mathcal{A} = (L, l_0, A, C, T, Inv, F, fm, \gamma)$ where $L, \ell_0, \mathcal{A}, C, T$, and Inv are defined as in TA (Definition 4.2.2), and F, fm, and γ are defined as follows:

- F is a finite set of features,
- $fm \in FE(F)$ is a feature model defined as a FE over features in F, and
- $\gamma : T \to FE(F)$ is a total function that assigns feature expressions to edges. Sometimes we shall write $l \xrightarrow[\varphi]{g,a,r} l'$ to express that $\gamma(l, g, a, r, l') = \varphi$ and that $(l, g, a, r, l') \in T$.

■

Now we can encode various models of similar systems into a single FTA by taken advantage of their similarities. Let us consider the payment controller introduced in Figure 4.3. Imagine we now want to model an SPL of payment controllers to capture systems with support for different payment methods: a) only credit card, b) only PayPal, and c) both. We can easily do this by representing the support for each method as features supported by the SPL. Figure 4.5 (top left) depicts such a FTA. The model is further explained in Example 4.2.2.

4.2.2 Example (FTA). Figure 4.5 exemplifies a simple FTA modeling an SPL of payment selection controllers. The base model corresponds to the TA depicted in Figure 4.3 and it is parameterized by two features *pp* and *cc*, standing for the support for paying by *PayPal* and by *credit card*, respectively. The transitions labelled with the action *paypal* and *creditcard* are only active when the corresponding feature is present, while the transition labelled with the action *pay* is active if at least one of the payment methods is present. The lower expression [$fm = pp \vee cc$] defines the *feature model*, i.e., how the features relate to each other. In this case at least one payment method must be supported. ◁

By selecting a desired combination of features, one can map an FTA into a concrete TA. Figure 4.5 (top right, and bottom) depicts the resulting TA obtained by projecting the payment controller into its three valid feature selections. The formal definition of projection follows. This definition has been slightly adapted from [47] to reflect the changes introduced in our modified definition of FTA.

4.2.5 Definition (FTA Projection). The *projection* of an FTA $\mathcal{A} = (L, l_0, A, C, T, Inv, F, fm, \gamma)$ over a set of features Fs, written $\mathcal{A}\downarrow_{Fs}$, is a timed automaton defined as follows:

$$\mathcal{A}\downarrow_{Fs} = (L, l_0, A, C, T', Inv)$$

where $T' = \{t \in T \mid Fs \models \gamma(t)\}$. ■

Givne a feature selection Fs, only transitions whose feature expression is satisfied by Fs remain present in the projection.

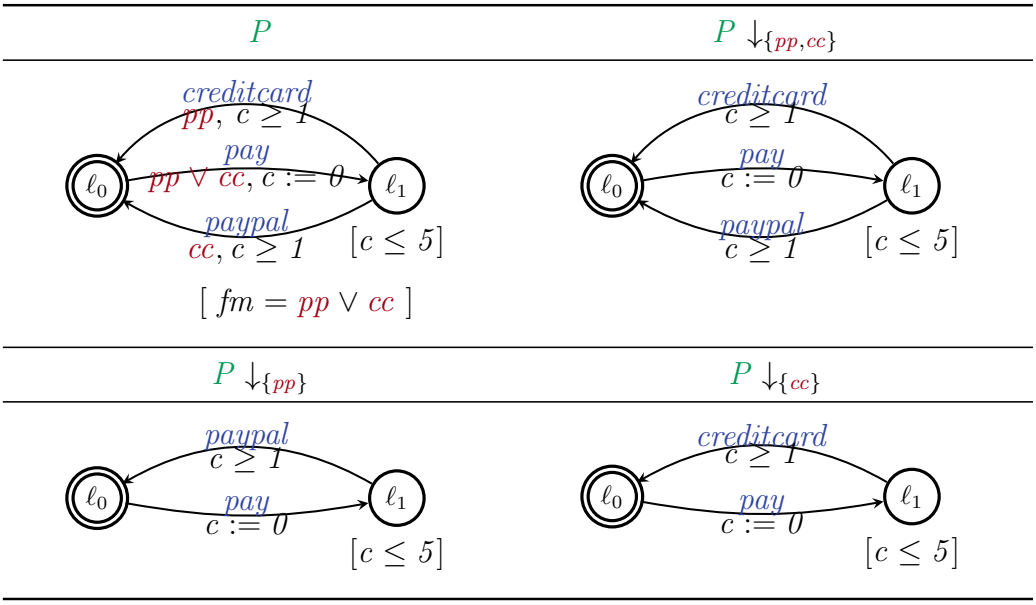


Figure 4.5: Example of a FTA, P, over the features pp and cc (top left), and its projections over its valid feature selections, cc and pp (top right), pp (bottom left), and cc (bottom right).

Cordy *et al.* proposes two alternative semantics for FTA [35]. As various transition systems, each corresponding to the underlying TS of each of its valid projections; and as an infinite *Featured Transition System* (FTS), similarly to how the semantics of TA is given in terms of TS.

Since a FTA models the behaviour of various TA, the semantics can be given as function that associates to each valid feature selection the semantics of the projected timed automata. This is given as follows.

4.2.6 Definition (Semantics of FTA in terms of TS). The *semantics of featured timed automaton* $\mathcal{A} = (L, l_0, A, C, T, Inv, F, fm, \gamma)$, is a function $[\![\mathcal{A}]\!]$ such that:

$$\forall\ Fs \in [\![fm]\!]^F\ \cdot\ [\![\mathcal{A}]\!](Fs) = [\![\mathcal{A} \downarrow_{Fs}]\!]$$

■

Alternatively, the semantics can be given in terms of FTS. An FTS extends Transition Systems with a set of features F, a feature model fm, and a total function γ that assigns FE to transitions. The formal definition of semantics of an FTA as an FTS follows. Notice that this definition of semantics has been adapted to incorporate the modifications over FTA adopted here, namely, no featured clock constraints and no application conditions.

4.2.7 Definition (Semantics of FTA as an FTS). The *semantics of a featured timed automaton* $\mathcal{A} = (L, l_0, A, C, E, Inv, F, fm, \gamma)$, is a featured transition system $[\![\mathcal{A}]\!] = (St, s_0, A, T, F, fm, \gamma')$, where:

- $St \subseteq L \times \mathbb{R}^C$ is the set of states consisting of the current location and clock valuation,
- $s_0 = \langle \ell_0, \eta_0 \rangle$ is the initial state,
- $T \subseteq St \times (A \uplus \mathbb{R}_{\geq 0}) \times St$ is the transition relation, sometimes written $s_1 \xrightarrow{\alpha} s_2$ to express that $(s_1, \alpha, s_2) \in T$, and
- $\gamma' \colon T \to FE(F)$ is a total function that assigns feature expressions to transitions in T, sometimes written $s_1 \xrightarrow[\varphi]{\alpha} s_2$ to express that $\gamma(s_1, \alpha, s_2) = \varphi$ for $(s_1, \alpha, s_2) \in T$.

The transition relation and γ are defined as follows, based on $\mathcal{A}$:

$$\langle \ell, \eta \rangle \xrightarrow[\top]{d} \langle \ell, \eta + d \rangle \quad \text{if} \ \ \eta \models Inv(\ell) \text{ and } (\eta + d) \models Inv(\ell), \ \text{ for } \ d \in \mathbb{R}_{\geq 0} \tag{4.3}$$

$$\langle \ell, \eta \rangle \xrightarrow[\varphi]{a} \langle \ell', \eta' \rangle \quad \text{if} \ \ \exists\ \ell \xrightarrow[\varphi]{g,a,r} \ell' \in E \ \text{ s.t. } \ \eta \models g, \ \ \eta \models Inv(l), \ \ \eta' = [r \mapsto 0]\eta, \ \text{ and } \ \eta' \models Inv(\ell') \tag{4.4}$$

■

Transitions in the resulting FTS are defined similarly to how transitions are defined for the underlying transition system of a TA (Definition 4.2.3). The main difference

is that now transitions are associated with a feature expression that determines the products (concrete TS) in which such transitions are present. Since delay transitions are independent from the variability, i.e., they depend only on the current clock valuation and on the invariant associated to the origin and target location, they are present in every product. Thus, their feature expression is always true ($\top$). The feature expression associated to an action transition, is simply the feature expression of the transition it represents in the corresponding FTA $\mathcal{A}$.

These alternative definitions of semantics enable two different ways to model check FTA: following a product by product approach, by analysing the underlying transition system of each TA projected; and following a variability aware approach that verifies properties over the entire family at once. The main characteristic of the second approach is that it takes advantage of the common behaviour of the various products in the family during verification, which can reduce significantly verification times. Of course, this might not be the case always, since in the case of very large SPLs it might be more efficient to take advantage of parallel processors to verify in parallel properties of different products or subsets of products of the SPL.

4.3 $\mathcal{R}$eo Coordination Language

Coordination of interacting and concurrent components can be done endogenously, or exogenously [9, 10]. In *endogenous* coordination, coordination is treated as a second-class concept, and it appears intertwine within the models of the components' computation. This hinders the study of properties of the components behaviour and the coordination protocol, as well as their maintainability and scalability. Examples of endogenous coordination includes: process algebras, share memory, and message passing, among others. In *exogenous* coordination, coordination is treated as a first-class concept, where the models of coordination protocols reside separately from the components they orchestrate. This facilitates anonymous communication of components, i.e., components have no knowledge of other components nor of the coordination protocols. Thus, it is possible to yield loosely-coupled and flexible systems whose components and coordination protocols can be easily study, maintain, and scale up. Examples of exogenous coordination include $\mathcal{R}$eo and to some extent, *Orc* [98].

Given that the nature of software product lines is to derive products from a common set of assets, including models, one would expect to maximize as much as possible (without hindering productivity) the reusability of such models. One step towards such direction is to take advantage of exogenous coordination to increase the reusability of both, the coordination protocols and the components, which are oblivious to other components and how they interact with each other.

Towards this goal, we focus on $\mathcal{R}$eo, a well known channel-based exogenous coordination language, introduced by Arbab [9], in order to understand exogenous coordination and how we can use this concept to model variable coordination protocols to coordinate families of services/components.

The language is based on the notion of channel composition [9]. In $\mathcal{R}$eo, complex coordinators, called *connectors*, are compositionally built out of simpler ones, called *channels*. Each channel has a set of input and output ports, and a well defined semantics of how data flows from the inputs to the outputs. Channels can be composed by connecting their ends through *nodes*.

These connectors can then be used to orchestrate how components in a component-based system interact. Since $\mathcal{R}$eo is simply a coordination formalism, it abstracts from what a component can be (models, processes, threads, etc.). Instead, it assumes that components have a set of input and output ports, such that an input port of a component can read from the output port of a connector to which it is connected to; while an output port of a component can write out data items into the input port of a connector to which it is connected to. Components themselves can be a connector or encapsulate various simpler components orchestrated through $\mathcal{R}$eo connectors.

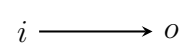
4.3.1 Primitive Channels

A channel consists of two ends and a constraint of how data flows through those ends. Channel ends can be of two types, *input* and *output*. An input end accepts data into its channel, and an output end dispenses data out of its channel. Each channel has associated an intuitive graphical notation that hints on how data flows through channels ends. As a consequence, it aids in the understanding of connectors, whose semantics is revealed by the resulting composed graphical notation.

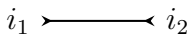
In the most basic notion of $\mathcal{R}$eo, constraints that specify the behaviour of a channel can be classified into *synchronisation constraints* and *dataflow constraints*. A synchronization constraint describes how the ends are synchronized on a particular step. Typically, it describes the atomicity, exclusion, order, and/or timing of the passage of data. For example, synchronous channels allow data to flow on both ends or neither end, while asynchronous channels allow data to flow on at most one of their ends. A dataflow constraint specifies constraints on the data that flows through the channel. Typically, it describes the content, the conditions for loss, and/or creation of data that passes through the ends of a channel. For example, a channel may specify that the data that flow through the input end must be exactly the same data that flows through the output end.

Channels are user defined, however there are a set of basic channels that can be used to built significantly complex connectors. We describe some of these channels

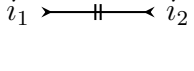
below.



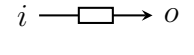
Sync. It synchronises both ends, i and o. It can only receive an input in i, if and only if, it can simultaneously send it through its output o.



SyncDrain. It is a synchronous drain. In synchronises both input ends, i_1 and i_2. In can only receive both inputs simultaneously, or not receive any input at all.



AsyncDrain. It is an asynchronous drain. It receives inputs in one input end at a time but never simultaneously.



FIFO$_1$. It introduces the notion of *storage* and *delay*. It can only receive an input i if it is not full. It is full if it has received an input before and has not been yet dispensed through the output o. By incorporating the notion of storage, it also incorporates the notion of delay, since after receiving an input time can pass before it is dispense. It is call FIFO$_1$ because it has storage for only *one* input at a time.

Notice that in the case of the drain channels, data received through the input ends is discarded, while in the case of the sync and fifo$_1$ the data item received must be the same data item dispensed through the output port.

It is worth mentioning that while $\mathcal{R}$eo channels have the notion of data items being written to input ports or being read out of output ports, for our purposes, instead of data items we focus on actions being executed/issued. For example, in the case of a sync, we say that an action connected to its input port can only be executed if the action connected to its output port can be executed simultaneously.

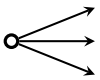
4.3.2 Nodes

Channels are composed by connecting their ends through nodes. There are three types of nodes in $\mathcal{R}$eo, namely, *input*, *output*, and *mixed* nodes[2]. The type of the node determines how synchronization between channels ends is conducted. A node is an

[2]Also referred as *source* node, *sink* node, and *mixed* node, respectively.

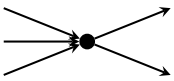
input, output, or mixed node if the channels ports that are connected to the node are all input ports, output ports, or a mixed of both ports, respectively.

Input and output nodes are called *boundary* nodes, i.e., they define the interface of a connector. Components can connect to these nodes and interact anonymously. At most one component can connect to one input or output node at a time. We use ○ to represent boundary nodes, and ● to represent mixed nodes. Mixed nodes remain hidden from the environment, i.e., from other components or connectors.



Input Node. A component can send data into an input node, only if the data can be send simultaneously through the input ports of all the outgoing channels. It is said that an input node acts as a *synchronous replicator*.

Output Node. A component connected to an output node and ready to receive data from it can do so, only if at least one of the channels connected to the node sends a data item. If more than one channel sends an item simultaneously, only one is selected non-deterministically. It is said that an output node acts as a *non-deterministic merger*.



Mixed Node. A mixed node non-deterministically synchronizes one of its coincident output ports with all of its coincident input ports. Data flow occurs only if all coincident input ports are ready to receive an item sent through one of the node's coincident output ports.

4.3.3 Connectors

We can now combine channels to build complex connectors. Two common and simple examples of composed connectors are the `merger` and the `replicator` connectors, which due to their common use and simplicity, they are typically treated as primitive connectors. Figure 4.6 shows their graphical representation. For simplicity, their boundary nodes are usually not represented.

A `replicator` consists of an input port and two output ports. It can receive an input only if it can simultaneously send it through all of its output ports.

A `merger` consists of two input ports and an output port. It synchronizes at most one of its input ports with its output port, i.e., if it receives both inputs simultaneously,

only one is synchronized with the output port. The choice is done non-deterministically. As mentioned before, an input and an output node act as a replicator and a merger, respectively.

Slightly more complex examples of connectors typically used are the `join` and `router` connector.

A `join` connector [12] consists of two input ports i_1 and i_2, an output port o, and two mixed nodes a and b. These nodes connect two syncs(i_1-a and i_2-b), a syncdrain (a-b), and two lossysyncs (a-o and b-o) as depicted in Figure 4.6b. This connector synchronizes both inputs with the output port. It can only receive both input simultaneously while sending the output without delay, or receive no input at all. Thus, the only possible behaviour is that the ends i_1-i_2-a-b-o synchronize and execution flows from i_1 and i_2 to o.

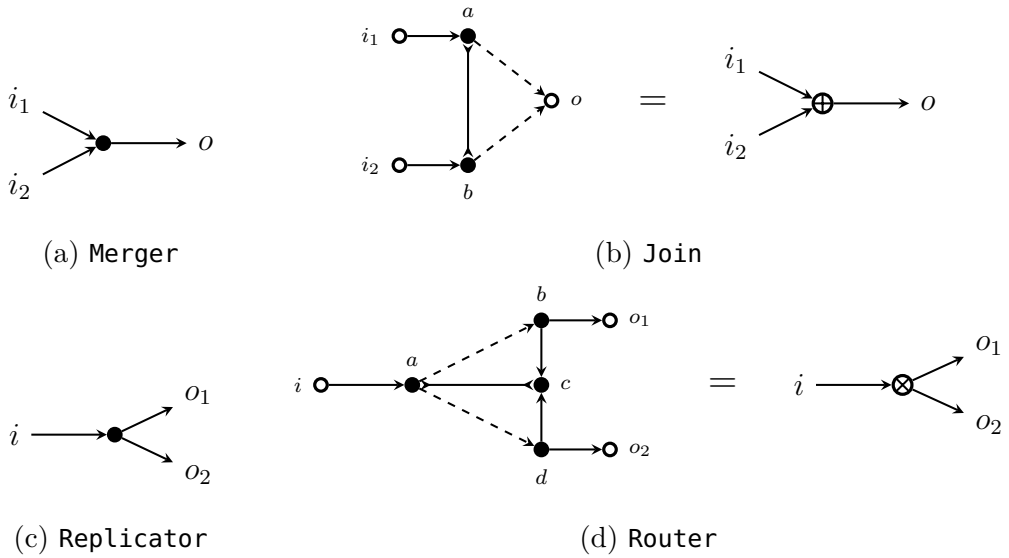
An exclusive router or just `router` [17] consists of an input port i, two output ports o_1 and o_2, and four hidden mixed nodes a, b, c and d. It is constructed by composing a sync (i-a), two lossysyncs (a-b and a-d), one syncdrain (a-c), and two replicators [3] (b-o_1-c and d-c-o_2), as shown in Figure 4.6d. It synchronizes the input port with at most one of its output ports. It can only receive an input if it can send it without delay through one of its output ports. If more than one output port is ready to receive the input, only one is selected non-deterministically. Thus, the possible behaviours are: 1) i-a-b-c-o_1 synchronize and execution flows from i to o_1, and 2) i-a-d-c-o_2 synchronize and execution flows from i to o_2.

Given that the join and router are common coordination mechanisms they are associated with a simpler graphical notation as shown next to each connector in Figure 4.6. Two new special nodes are introduced, $\oplus$ and $\otimes$, to represent both connectors. The former is connected to exactly two input ports and one input port, while the latter is connected to exactly one output port and two input ports. In practice, it is common to use n-output replicators and routers, and n-input mergers and joins, with $n = 2, 3, \ldots$, and they can easily be built by composing various connectors of the same kind.

4.3.4 $\mathcal{R}$eo Semantics

Depending on the nature of the components whose actions need to be coordinated, there are various formalisms to express the semantics of $\mathcal{R}$eo connectors. In fact, there are more than 30 semantics for $\mathcal{R}$eo [91], including operational models, such as constraint automata [17] with variants too capture the nature of the components being coordinated (timed [11], probabilistic [15], etc.); colouring models [33] to capture how

[3] For simplification, both replicators are shown as a mixed node, which since it has only one coincident output port, it acts as input node, i.e., as a synchronous replicator



(a) `Merger` (b) `Join`

(c) `Replicator` (d) `Router`

Figure 4.6: Examples of composed $\mathcal{R}$eo connectors and their simplified graphical notation.

data flow between nodes; and coalgebraic models, e.g., based on timed data streams (TDS) [13], to capture which and when data flows in a given node, among others.

In the context of SPLs, components can have variable interfaces, i.e. interfaces are not present in every product. To orchestrate this kind of components, protocols need to adapt to the presence and absence of such interfaces. To do this, we propose to model $\mathcal{R}$eo protocols using the compositional formalism presented in Chapter 5. We discuss this experience in Chapter 7.

Chapter 5

Compositional Modelling of Software Product Lines

This chapter discusses the compositional modelling of families of systems, in particular families of real time systems through formalisms based on automata theory. In Section 4.2 we discussed an existing *fine-grain* formalism called featured timed automata used to represent families of real timed systems in a single model. In this chapter we take a step forward and propose *Interface Featured Timed Automata* (IFTA), a *compositional* formalism based on both FTA and on notions of interface and I/O automata [50, 52]. This formalism enriches FTA with:

- *variable interfaces* by means of associated feature expressions that restrict the way multiple automata interact,
- *multi-action labels* associated to transitions that simplify the design of synchronous coordination, and
- a *compositional semantics* that takes into account the variability of the composed models.

The compositional semantics is the main contribution of this formalism. In addition to enable verifying behavioural properties of the entire family, as one can also do in FTA, it allows us to reason about variability when composing models that have their own feature model. For example, we can verify if the composed model realizes all feature dependencies specified by an expected feature model, or if it does not introduce new dependencies not specified by the expected feature model. In other words, it address questions like: does the composed model derives the expected set of product and only those?. Furthermore, we can study how components interact at an architec-

tural level by visualizing how components are connected at the family level and at the product level.

Chapter organization. First, in Section 5.1 we motivate with examples some aspects to be considered when composing families of systems. In Section 5.2, we present the formal definition of IFTA, its semantics, and operations. In Section 5.3 we discuss some related work, and wrap up with some discussion in Section 5.4.

5.1 Motivation

There are two aspects that need to be considered when modelling families of (distributed) services in a compositional manner. On the one hand, it is necessary to model mechanisms to coordinate variable services that need to adapt correctly to the presence or absence of such services. This task can quickly become cumbersome and error prone if done manually. On the other hand, when composing variable services, we also need to compose their feature models, which can be independent or have common features. In the following sections, we propose two simple scenarios to illustrate these aspects. To motivate the need for compositional modelling and semantics, we model these components as FTA.

5.1.1 Coordinating Variable Services

Let us consider a simple scenario where we have a system which may support remote requests to two databases, DB_1 and DB_2, by a given user U. This may represent a typical situation when a given authority must consult external databases to verify information of an applicant, such as an existing criminal record, traffic violations, and tax crimes, among others.

The general behaviour of the system is as follows. Typically, if communication with both databases is supported, both are consulted, otherwise only the available database is consulted. The user must wait for all responses before being able to do another request. If none is supported, the user should not be able to make requests. To guarantee model decouplement and separation of concerns, we decompose the system into various components as depicted in Figure 5.1. Furthermore, in alignment with the principles of exogenous coordination, we distinguish between the main building blocks of the system, i.e. the databases and the user, and the *gluing code* that puts them together, i.e. a two variable coordination mechanism: Req, which models how to make requests to both databases depending on their presence, and Res, which models how to merge the responses from the available databases before presenting them to the user.

We describe each of these components below. Notice that for simplicity we avoided introducing time constraints in the examples. In reality each database may have its own processing time, and the user might specify a time-out for responses.

Notation. In order to model simultaneity (synchronous steps), we adopt the notation C associated to a location as used in UPPAAL. A location with C is a *committed location*, meaning time can not pass in that location. When an automaton in a network is in a committed location, only outgoing transitions from there are enabled. This allows to model a sequence of actions executing simultaneously. For example, in the FTA *Req* from Figure 5.1, the actions in the sequence

$$\ell_0 \xrightarrow[db1 \vee db2]{request} \underset{\mathsf{C}}{\ell_1} \xrightarrow[db1 \wedge \neg db2]{db1in} \ell_0$$

execute simultaneously because time can not pass in location ℓ_1. This notation is a simplification for the sequence

$$\ell_0 \xrightarrow[db1 \vee db2]{request, c} \underset{c \leq 0}{\ell_1} \xrightarrow[db1 \wedge \neg db2]{db1in} \ell_0$$

where a clock c is reset to 0 in all incoming edges to ℓ_1 and an invariant $c \leq 0$ is associated to ℓ_1.

User (*U*). This component has two actions, *request* and *response*, which stand for the support to consult external databases and receive their responses, respectively. The presence of both actions depends on the presence of at least one database, as indicated by the feature expression $db1 \vee db2$ associated to both edges. The user can simply make a request and wait for the response.

Databases (DB_1 and DB_2). Both databases have two actions, *dbNin* and *dbNout*, which stand for the support to consult database N, for $N = 1, 2$, and issue the corresponding result, respectively. Two features, *db1* and *db2*, represent the support for database DB_1 and DB_2, respectively.

Request (*Req*). It has three actions, *request*, *db1in*, and *db2in*, which synchronize [1] with the equally named actions in *U*, DB_1 and DB_2, respectively. This component

[1] To recall from Section 4.2.1, when automata are composed in parallel, they can synchronized over shared action names, meaning that two automata are in a state where both can take an outgoing transition labelled with the same action name, they transition together simultaneously, i.e., both move to a new state. They can not transition independently over transitions with shared actions.

coordinates how to consult the available databases if a *request* is issued by U. If only one database is present, it consults the corresponding database. For example if only DB_1 is present, only the edge $\ell_1 \xrightarrow[\mathsf{C}]{\substack{db1in \\ db1 \wedge \neg db2}} \ell_0$ will be present from location ℓ_1. If both are present, they are called simultaneously, i.e. *db1in* and *db2in* must be called at the same time. This is model as a sequence of actions with committed locations. To avoid imposing an order, both possible sequence are modeled, i.e. $\ell_1 \xrightarrow{\substack{db1in \\ db1 \wedge db2}} \ell_2 \xrightarrow[\mathsf{C}]{\substack{db2in \\ db1 \wedge db2}} \ell_0$ and $\ell_1 \xrightarrow{\substack{db2in \\ db1 \wedge db2}} \ell_3 \xrightarrow[\mathsf{C}]{\substack{db1in \\ db1 \wedge db2}} \ell_0$. If neither database is available, the entire automaton is not present, i.e., there are no transitions in the resulting projection.

Response (*Res*). It has three actions, *response*, *db1out*, and *db2out*, which synchronize with the same action names in U, DB_1 and DB_2, respectively. This component merges the responses from the available databases before sending them to the user. If only one database is available, it waits for the corresponding response. For example, if only DB_1 is present, only the sequence $\ell_0 \xrightarrow{\substack{db1out \\ db1 \wedge \neg db2}} \ell_2 \xrightarrow[\mathsf{C}]{\substack{response \\ db1 \vee db2}} \ell_0$ will be present. If both are present, each database can have its own timing and results may be ready in any possible order. For example, the sequence $\ell_0 \xrightarrow{\substack{db1out \\ db1 \wedge db2}} \ell_3 \xrightarrow{\substack{db2out \\ db1 \wedge db2}} \ell_2$ captures database DB_1 returning a result before DB_2. If neither database is available, the entire automaton is not present.

Note that both coordinators designed in Figure 5.1 correspond to two $\mathcal{R}$eo connectors, namely, a *replicator* (*Req*) and a *join* (*Res*), here modelled with variable behaviour.

As mentioned at the beginning of Section 5.1, modelling families of services and coordinating how they are integrated can become cumbersome and error prone. In particular, we recognize two potential issues, as follows.

Modeling variable coordinators. The difficulty in manually modelling such mechanisms is twofold. First, the variability of these mechanisms depends on the variability of the services they orchestrate. Modelling the feature expressions associated to each transition is done manually taking into consideration this characteristic. This is error prone, it can lead to erroneous behaviour such as edges that can never be taken or that can be taken when they should not be present; and it is inefficient when modelling complex coordinators. For example, in the FTA *Req*, if the feature expression associated to the transition $\ell_1 \xrightarrow{db1in} \ell_0$ is erroneously set to $db1 \wedge db2$ instead of $db1 \wedge \neg db2$, the model allows calling only database DB_1 when both databases should be called. Second, the aim of these mechanisms is to coordinate how multiple actions synchronize

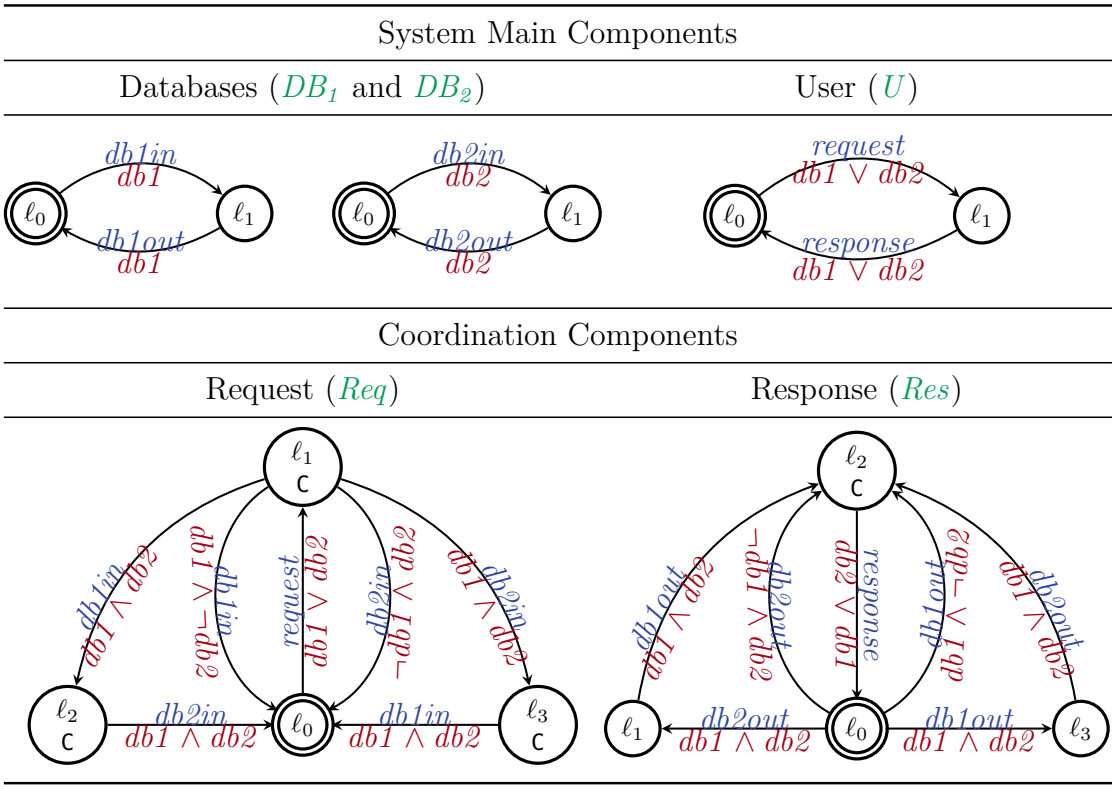


Figure 5.1: An example of a network of FTA modelling a family of systems which can make remote requests to available databases.

and wait on each other. In many cases, this requires modelling various alternative behaviours, i.e., alternative orders in which actions can execute. Identifying all cases manually can be error prone and inefficient. For example, in the FTA Res, if we fail to identify that DB_2 may finished before DB_1, we may end up with a deadlock if this were to happen. Although these issues may be identified through model checking, in some cases, it depends on the formulation of the correct properties. Then, it may be desirable to prevent this type of errors by design.

Evolving variable coordinators. Introducing changes to manually designed coordinators, requires to redesign the entire coordinator and calculate manually the new feature expressions associated to transitions. This is time-consuming, and it can become cumbersome and error prone quickly. In our example, if in the future a third database is added, it will be necessary to redesign both coordinators entirely and cal-

culate manually the new feature expressions. In the case of *Req*, such an automaton for three databases, has 11 locations and 25 transitions.

5.1.2 Composing Variable Services

Let us now consider a different scenario, where we have a system which may support online payment transactions and, if available, send email notifications if the payment is successful. The system is divided into a payment component, P and a notification component N.

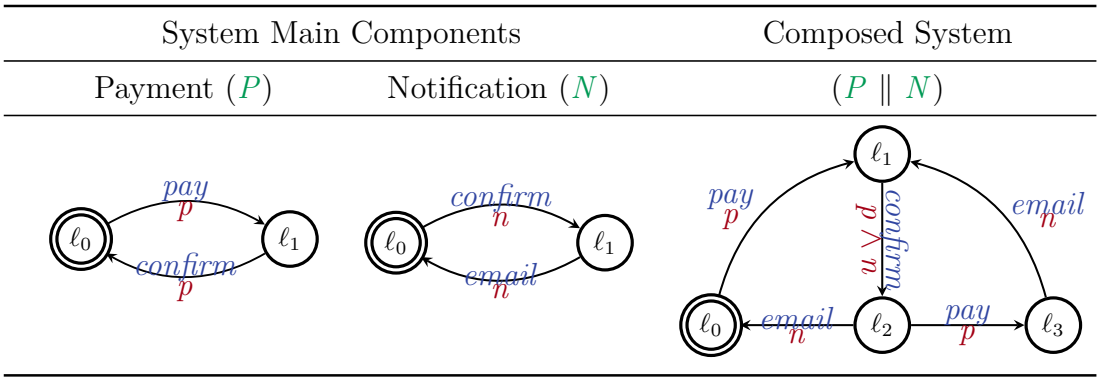


Figure 5.2: Example of a network of FTA modelling a family of payment systems which may send email confirmations

Payment (P). The service receives a payment request, *pay*, and emits a confirmation event, *confirm*, if the payment was successful. A feature p represents the support for the payment service, which can be present or absent, i.e., $fm_P = \top$.

Notification (N). The services receives a confirmation notification, *confirm*, which synchronizes with the same action name from P, after which it sends an email notification, *email*. A feature n represents the support for the email notification service, which can be present or absent, i.e., $fm_N = \top$.

Composed System ($P \parallel N$). The composed service results from the parallel composition of P and N, where both automata can interleave, i.e. they can jump independently over non-shared actions, or synchronize over the action *confirm*, i.e. they jump simultaneously over this action. The feature expression of a synchronized transition

is the logical conjunction of the feature expressions of independent transitions being synchronized.

The main issue is how the new feature model should be defined. If we simple express the composed feature model as the logical conjunction of both feature models, then $fm_{P \| N} = \top$. However, this results in a undesirable behaviour when only feature p is present. In the projected automaton $(P \parallel N) \downarrow_p$, after a payment is requested, the service can no longer emit a confirmation in case of success. Thus, we need to impose additional restrictions to the composed feature model. Intuitively, this restriction should impose that the synchronized actions should always be present or absent together. In Section 5.2.3 we present a compositional semantics that takes this into consideration.

5.2 Interface Featured Timed Automata

This section introduces the proposed compositional formalism. We present IFTA syntax in Section 5.2.1, discuss its semantics in Section 5.2.2, and how IFTA are composed in Section 5.2.3. Finally, in Section 5.2.4 we propose a notion of IFTA equivalence by means of a bisimulation relation, and use it in Section 5.2.5 to study properties of IFTA composition.

5.2.1 Syntax

Interface Featured Timed Automata extends FTA mainly in three ways. In this section we discuss the first two extensions, namely, interfaces and multi-action labels. The compositional semantics extension is discussed in Section 5.2.3.

First, *synchronizing actions* are lifted into the interface of the automaton, representing the actions through which an automaton can communicate with the environment, namely, other automata when composed in parallel. A synchronizing action a, can be an *input* or an *output* action, written $a?$ and $a!$, respectively. An input action represents an event expected from the environment, while an output action represents an event sent to the environment. We say an IFTA is *grounded* if it has a feature expression associated with each interface action. This association is done only once. The feature expression associated is *inferred* from the definition of the automaton. Intuitively, it represents the valid set of products in which an action was designed to be present.

Second, transitions can be labelled with a set of actions instead of only one action to model simultaneous execution of actions. This simplifies significantly the design and construction of complex synchronous coordination mechanism, as we will see in

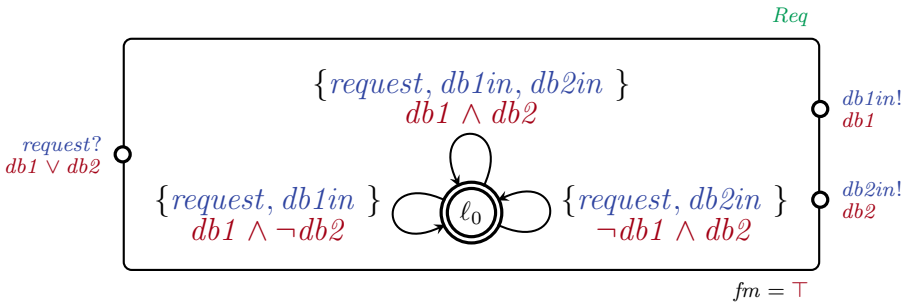


Figure 5.3: A *grounded* IFTA corresponding to the FTA *Req* (Figure 5.1).

more detail in Chapter 7. A transition is enabled only when all of its actions can be taken at the same time and if the current clock valuation satisfies its clock constraint. Synchronizing actions can be taken when their dual actions on neighbour automaton are on an enabled transition. The dual of an interface action a, noted $\overline{a}$, is defined as $\overline{a!} = a?$ and $\overline{a?} = a!$. In addition to synchronizing actions, there are internal actions (not visible to the environment), which can always be taken if the transition is enabled.

Figure 5.3 exemplifies how the automaton *Req* from Figure 5.1 is modelled as an IFTA by identifying input and output actions and lifting them into the interface of the automaton, depicted with ○, and by modelling simultaneous execution of actions with multi-action labels associated to transitions.

Now we formalize the definition of IFTA, interface, feature expression of an action, and grounded IFTA.

5.2.1 Definition (Interface Featured Timed Automata). An *interface featured timed automaton* is a tuple $\mathcal{A} = (L, l_0, A, C, T, Inv, F, fm, \gamma)$ where L, l_0, C, Inv, F, fm, and γ are defined as in FTA (Definition 4.2.4), and A and T are defined as follows:

- $A = I \uplus O \uplus H$ is a finite set of actions, where I is a set of input ports, O is a set of output ports, and H is a set of hidden[2] (internal) actions, and
- $T \subseteq L \times CC(C) \times \mathbf{2}^A \times \mathbf{2}^C \times L$ is a finite set of transitions, now labelled with a set of actions instead of a single action.

■

Notation. When not clear from the context, we will use $L_{\mathcal{A}}, l_{0_{\mathcal{A}}}, A_{\mathcal{A}}, \ldots$ to refer to the elements of an IFTA $\mathcal{A}$. When using automata names with subscripts such

[2]Typically $H = \{\tau\}$

as $\mathcal{A}_1, \mathcal{A}_2, \ldots$, we will simply use $L_1, L_2, l_{0_1}, l_{0_2}, \ldots$. For simplicity, sometimes we write $l \xrightarrow{g,\omega,r}_{\mathcal{A}} l'$ instead of $(l, g, \omega, r, l') \in E_{\mathcal{A}}$, and use $l \xrightarrow[\varphi]{g,\omega,r}_{\mathcal{A}} l'$ to express that $(l, g, \omega, r, l') \in E_{\mathcal{A}}$ and $\gamma_{\mathcal{A}}(l, g, \omega, r, l') = \varphi$.

The *interface* of an IFTA $\mathcal{A}$ is the set $\mathbb{P}_{\mathcal{A}} = I_{\mathcal{A}} \uplus O_{\mathcal{A}}$ of all input and output ports of $\mathcal{A}$. Given a port $p \in \mathbb{P}_{\mathcal{A}}$ we write p instead of $\{p\}$ when clear from context.

At this point, the definition of IFTA only incorporates the notion of feature expressions associated to transitions through function γ, but does not incorporate the notion of feature expressions associated to interfaces. Before doing this, we define the notion of feature expression of an action. Given an IFTA $\mathcal{A}$, it is possible to infer for each action $a \in A_{\mathcal{A}}$ a feature expression based on the feature expressions of the edges in which a appears. Intuitively, this feature expression determines the set of products requiring a. The formal definition follows.

5.2.2 Definition (Feature Expression of an Action). Given an IFTA $\mathcal{A}$, the inferred feature expression of any action $a \in A_{\mathcal{A}}$ is the disjunction of all the feature expressions associated to transitions where a appears, defined as

$$\widehat{\Gamma}_{\mathcal{A}}(a) = \bigvee \{\gamma_{\mathcal{A}}(l \xrightarrow{g,\omega,r}_{\mathcal{A}} l') \mid a \in \omega\} \qquad \text{(FE of an action)}$$

■

Now we can associate feature expressions to the actions of an IFTA. In order to do this, we incorporate a new function Γ to the definition of an IFTA $\mathcal{A}$, qualifying the result as *grounded*. Thus, a *grounded* IFTA is given by $\mathcal{A} = (L_{\mathcal{A}}, l_{0_{\mathcal{A}}}, A_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}}, Inv_{\mathcal{A}}, F_{\mathcal{A}}, fm_{\mathcal{A}}, \gamma_{\mathcal{A}}, \Gamma)$, where $\Gamma : A_{\mathcal{A}} \to FE(F_{\mathcal{A}})$ is a total function that assigns a feature expression to each action of $\mathcal{A}$, and is defined as $\widehat{\Gamma}_{\mathcal{A}}(a)$ for all $a \in A_{\mathcal{A}}$ at the moment of the grounding. By doing this association only once, we are *fixing* the feature expression associated to each action, such that it represents the set of products where each action was originally design to be present in.

The need for this function and for fixing it instead of using directly $\widehat{\Gamma}$ has to do with the way we define the composition of IFTA and the properties that we expect from it. We discuss this in Section 5.2.3.

We shall work only with grounded IFTA from now own.

5.2.2 Operational Semantics

We define the semantics of IFTA in terms of *Interface Feature Transition Systems* (IFTS), and define an IFTS as a featured transition system with an interface, multi-action labels, and feature expressions associated to actions.

5.2.3 Definition (Interface Featured Transition System). An IFTS is a tuple $S = (St, s_0, A, T, , F, fm, \gamma, \Gamma)$, where St is a set of states, s_0 is the initial state, $A = I \uplus O \uplus H$ is the set of actions where I, O, and H are the set of input, output, and hidden actions, respectively, $T \subseteq St \times (\mathbf{2}^A \uplus \mathbb{R}_{\geq 0}) \times St$ is the transition relation, F is a set of features, fm is the feature model, $\gamma : T \to FE(F)$, is a total function that assigns feature expressions to transitions, and $\Gamma : A \to FE(F)$, is a total function that assigns feature expressions to actions. ■

Notation. As before, when not clear from the context, we will use $St_S, s_{0_S}, A_S, \ldots$ to refer to the elements of a transition system S.

5.2.4 Definition (Semantics of an IFTA as an IFTS). The semantics of a grounded IFTA $\mathcal{A} = (L, l_0, A, C, T, Inv, F, fm, \gamma, \Gamma)$, is an interface featured transitions system

$$[\![\mathcal{A}]\!] = (St, s_0, A, T', F, fm, \gamma', \Gamma)$$

where

- $St \subseteq L \times \mathbb{R}^C$ is the set of states, where in a state $\langle \ell, \eta \rangle \in St$, ℓ is a location, and η is a clock valuation,
- $s_0 = \langle \ell_0, \eta_0 \rangle$ is the initial state,
- $T' \subseteq St \times (\mathbf{2}^A \uplus \mathbb{R}_{\geq 0}) \times St$ is the transition relation, and
- $\gamma' : T' \to FE(F)$ is the total function that assigns feature expressions to transitions in T'.

The transition relation and γ are defined as follows,

$$\langle \ell, \eta \rangle \xrightarrow[\top]{d} \langle \ell, \eta + d \rangle \text{ if } \eta \models Inv(\ell) \text{ and } (\eta + d) \models Inv(\ell), \text{ for } d \in \mathbb{R}_{\geq 0} \tag{5.1}$$

$$\langle \ell, \eta \rangle \xrightarrow[\varphi]{\omega} \langle \ell', \eta' \rangle \quad \text{if } \exists\ \ell \xrightarrow[\varphi]{g,\omega,r} \ell' \in T \text{ s.t. } \eta \models g, \ \eta \models Inv(l),\ \eta' = [r \mapsto 0]\eta, \text{ and } \eta' \models Inv(\ell') \tag{5.2}$$

■

5.2.3 Composition

In this section we discuss the third extension proposed with respect to FTA, namely, the compositional semantics for IFTA that takes into account the variability models of the IFTA being composed.

Informally, two IFTA can be composed by combining their feature models and linking interfaces, imposing new restrictions over them. The composition is built on top of two simpler operations: *product* and *synchronization*. The product operation for IFTA, unlike the classical product of timed automata, is defined over grounded IFTA with disjoint sets of actions and clocks, performing their transitions in an interleaving or synchronous-step fashion. The formal definition of product follows.

5.2.5 Definition (Product of IFTA). Let $\mathcal{A}_1$ and $\mathcal{A}_2$, be two different grounded IFTA with disjoint actions and clocks. The product of $\mathcal{A}_1$ and $\mathcal{A}_2$, is a new IFTA

$$\mathcal{A}_1 \times \mathcal{A}_2 = (L_1 \times L_2, \ell_{0_1} \times \ell_{0_2}, A, C_1 \cup C_2, F_1 \cup F_2, T, Inv, fm_1 \wedge fm_2, \gamma, \Gamma)$$

where A, T, Inv, γ and Γ are defined as follows:

- $A = I \uplus O \uplus H$, where $I = I_1 \cup I_2$, $O = O_1 \cup O_2$, and $H = H_1 \cup H_2$.
- T and γ are defined by the rules below, for any $\omega_1 \subseteq A_1$, $\omega_2 \subseteq A_2$.

$$\frac{\ell_1 \xrightarrow[\varphi_1]{g_1,\omega_1,r_1}_1 \ell_1'}{\langle \ell_1, \ell_2 \rangle \xrightarrow[\varphi_1]{g_1,\omega_1,r_1} \langle \ell_1', \ell_2 \rangle} \qquad \frac{\ell_2 \xrightarrow[\varphi_2]{g_2,\omega_2,r_2}_2 \ell_2'}{\langle \ell_1, \ell_2 \rangle \xrightarrow[\varphi_2]{g_2,\omega_2,r_2} \langle \ell_1, \ell_2' \rangle}$$

$$\frac{\ell_1 \xrightarrow[\varphi_1]{g_1,\omega_1,r_1}_1 \ell_1' \quad \ell_2 \xrightarrow[\varphi_2]{g_2,\omega_2,r_2}_2 \ell_2'}{\langle \ell_1, \ell_2 \rangle \xrightarrow[\varphi_1 \wedge \varphi_2]{g_1 \wedge g_2, \omega_1 \cup \omega_2, r_1 \cup r_2} \langle \ell_1', \ell_2' \rangle}$$

- $Inv(\ell_1, \ell_2) = Inv_1(\ell_1) \wedge Inv_2(\ell_2)$.
- $\forall\ a \in A\ \cdot\ \Gamma(a) = \Gamma_i(a)$ if $a \in A_i$, for $i = 1, 2$.

■

Both top transitions represent the interleaving of both automata, i.e., either $\mathcal{A}_1$ takes a transition (top left) or $\mathcal{A}_2$ takes it (top right), but not both. The bottom transition represents the synchronous execution of transitions from $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e., when both take a transition a the same time [3]. In the first two cases, the feature expression of the existing transition in $\mathcal{A}_i$, for $i \in \{1, 2\}$, is carried to the new transition in the composed automata. In the last case, the feature expression of the synchronous transition results from the conjunction of the feature expression associated to the transitions

[3] In the regular notion of product of timed automata, this is only possible over transitions with the same shared action. However, in this case the automata do not share actions, instead we postponed the synchronization to a later stage, by explicitly linking ports.

being synchronised. In the case of Γ, since the intention is to maintain the original set of products for which an action a was defined to be present in, we define Γ as Γ_i for actions in A_i.

The *synchronisation* operation over an IFTA $\mathcal{A}$ connects and synchronises two actions a and b in $A_{\mathcal{A}}$. The resulting automaton has transitions without neither a and b, nor both a and b.

5.2.6 Definition (Synchronisation). Given a grounded IFTA $\mathcal{A} = (L, \ell_0, A, C, F, T, Inv, fm, \gamma, \Gamma)$ and two actions $a, b \in A$, the synchronisation of a and b is a new IFTA

$$\Delta_{a,b}(\mathcal{A}) = (L, \ell_0, A', C, F, T', Inv, fm', \gamma, \Gamma)$$

where A', E' and fm' are defined as follows:

- $A = I' \uplus O' \uplus H'$, where $I' = I \setminus \{a.b\}$, $O' = O \setminus \{a.b\}$, and $H' = H \cup \{a.b\}$.
- $T' = \{\ell \xrightarrow{g,\omega,r} \ell' \in T \mid a \notin \omega \text{ and } b \notin \omega\} \cup$
  $\{\ell \xrightarrow{g,\omega\setminus\{a,b\},r} \ell' \mid \ell \xrightarrow{g,\omega,r} \ell' \in T \text{ and } a \in \omega \text{ and } b \in \omega\}$
- $fm' = fm \wedge (\Gamma_{\mathcal{A}}(a) \leftrightarrow \Gamma_{\mathcal{A}}(b))$.

■

The resulting feature model imposes new restrictions over the set of features based on the actions being synchronised. Intuitively, if two actions a and b are synchronised, they depend on each other. Thus, we require that they should both be present or both absent in any valid set of features $(\Gamma_{\mathcal{A}}(a) \leftrightarrow \Gamma_{\mathcal{A}}(b))$.

Together, product and synchronisation can be used to obtain in a *compositional* way a complex IFTA built out of more simple ones. We define the composition of two IFTA as their product, followed by the explicit binding of actions through synchronization. The composition is defined for interface actions synchronized on an input-output fashion only.

5.2.7 Definition (Composition of IFTA). Given two grounded IFTA, $\mathcal{A}_1$ and $\mathcal{A}_2$, with disjoint set of actions, and clocks; and a possibly empty set of bindings $\{(a_1, b_1), \ldots, (a_n, b_n)\}$, such that, for each pair a_i and b_i, $1 \leq i \leq n$, we have that

$$(a_i, b_i) \in I_1 \times O_2 \text{ or } (a_i, b_i) \in O_1 \times I_2 \qquad (io\text{-only})$$

then their composition is a new grounded IFTA defined as follows

$$\mathcal{A}_1 \bowtie_{(a_1,b_1),\ldots,(a_n,b_n)} \mathcal{A}_2 = \Delta_{a_1,b_1} \ldots \Delta_{a_n,b_n}(\mathcal{A}_1 \times \mathcal{A}_2)$$

■

Figure 5.4 exemplifies the composition of a variable router [4] R (top left) and a simple PayPal component (top right). The variable router corresponds to the $\mathcal{R}$eo router in Section 4.3.3, except that in this case it encodes three possible connectors: a typical router when all ports are present; and two different sync when either only i and o_1 are present, or i and o_2 are. It is also possible for all ports to be absent in which case the entire connector is absent. In fact, the router captures the behaviour of the payment selection method in Figure 4.5 if its invariant in location l_1 is $c \leq 0$. The PayPal component models an oversimplified PayPal payment method that after a payment is made it sends either a confirmation or an error notification in no more than 10 units of time. For simplification, only one output of the router is linked with a payment method, in this case PayPal. The other output could be connected with other possible payment method such as a credit card, while the input could be connected to any service that requires a payment. Then, whenever a payment is required, the router can link to all possible payment methods. The composition is done by linking the ports o_1 with $paypal$. The resulting IFTA combines the feature models of both IFTA, imposing additional restrictions given by the linked ports, in this case, $(f_i \wedge f_{o_1}) \leftrightarrow pp$, which imposes that o_1 is present if and only if, $paypal$ is present as well. The availability of each port is given by the feature expression associated to it. In the composed IFTA, transitions with linked actions are fired together: $\langle \ell_2, \ell_0 \rangle \xrightarrow{i} \langle \ell_2, \ell_1 \rangle$ is the joint transition of $\ell_2 \xrightarrow{i,o_1} \ell_2$ and $\ell_0 \xrightarrow{paypal} \ell_1$; while transitions with non-linked actions can execute together or independently.

By allowing each IFTA to have its own feature model and taking into account variability during composition, we can reason about how composing families of timed automata in parallel affects the presence of interfaces and the variability of the composed system. In particular, the feature model of the composed systems specifies what products can be derived, and for each product, we can seen its interfaces and connections. This is illustrated in Section 7.2.

Composition and inferred feature expressions

Because we define composition as the product followed by the synchronization, the product will produce transitions that are later *cut* by the synchronization when linking actions. This has the undesired effect that the order in which actions are linked, and therefore the order in which transitions are cut by the synchronization operation affects the inferred feature expression of an action.

If we were to use $\widehat{\Gamma}$ instead of Γ, synchronization would not be commutative, since the final feature expression of an action could differ depending on the order of syn-

[4] Variable coordination is discussed in Chapter 7

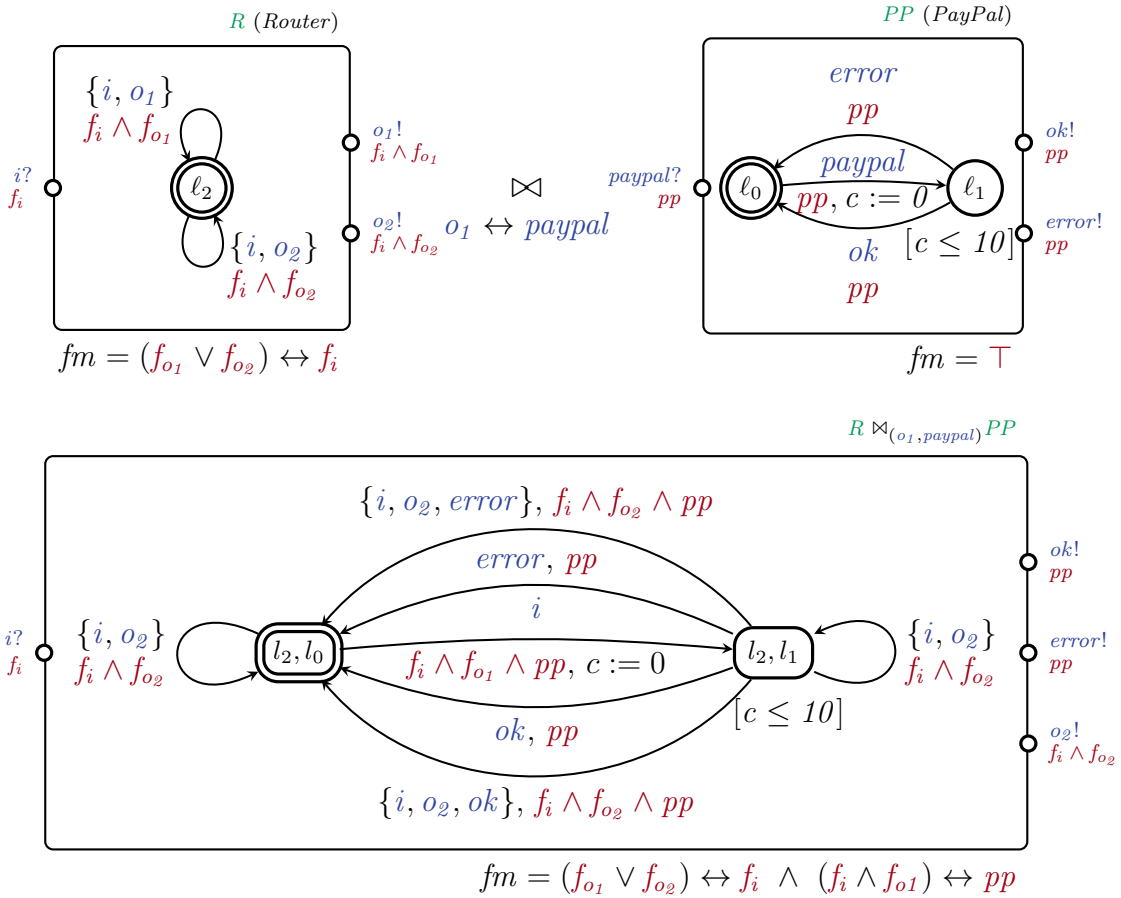


Figure 5.4: Composition of a Router IFTA (top left) with a PayPal IFTA (top right) by binding ports o_1 and *paypal*, yielding the IFTA below.

chronization. By *fixing* the feature expression of an action before composition, we avoid this issue and the synchronization remains commutative.

As a consequence, when we look at the feature expression of an action, it is necessary to look at it with respect to the feature model of the automaton. For example, if we have $\Gamma_{\mathcal{A}}(a) = \top$ for some action a and automaton $\mathcal{A}$, then a is present in all products *allowed* by the feature model of $\mathcal{A}$. This is, it may no longer be true that a is always present.

5.2.4 Equivalence

When dealing with timed automata, equivalence relations are defined over their underlying infinite transition systems, which represent the actual semantic behaviour of the finite automata. Similarly, we can define an equivalence relation for IFTA over their underlying transition systems. Furthermore, this relation, can be variability aware, meaning it takes advantage of the encoding of various systems in a single model, i.e. it is defined over IFTS; or it can be done in a product by product approach, i.e., over the underlying transitions system of each valid TA encoded in the family.

We define equivalence in terms of a bisimulation relation. Bisimulation compares two states, and requires that each of them mimics the other. Equivalence relations can be *state-based* or *action-based*, and can have different levels of strictness.

State-based approaches relate two transition systems by considering labels associated to states, called *atomic propositions*. Action-based approaches relate two transition systems by considering action labels associated to transitions. We adopt the latter since we are focusing on relating systems that define their behaviour in terms of actions, rather than on properties valid in a given state. Although, as explained in Section 4.2, the original definition of FTA uses atomic propositions, IFTA only uses action labels associated to transitions.

The level of strictness has to do with the actions considered by the relation. This section proposes an equivalence relation in terms of *strong bisimulation* ($\sim$). Strong bisimulation considers two transition systems to be bisimilar only if they are bisimilar for all their actions, including internal actions. This allows us to compare two IFTA modelling the same system and study properties of the operations defined in Section 5.2.3. Another possibility, not addressed here, is to define a more relaxed notion of bisimulation, commonly refer as *observational bisimulation* [74], which considers only the observable behaviour, i.e., input and output actions.

Cordy [46] proposes as well two types of simulation relations for FTS, one that separates variability from behaviour and it is defined in terms of TS, and one that takes advantage of similarities of the systems, defined in terms of FTS. In both cases, the relation is state-based.

Product by Product

In a product-by-product approach, bisimulation is defined over the underlying transition system of each valid TA that can be projected from an IFTA. Since an IFTA has an interface and multi-action transitions, we define a new kind of transition system, called Interface Transition System (ITS), which can be seen as an IFTS without variability.

5.2.8 Definition (Interface Transition System). An ITS is a tuple $S = (St, s_0, A, T)$, where St is the set of states, s_0 is the initial state, $A = I \uplus O \uplus H$ is the set of actions where I, O, and H are the set of input, output, and hidden actions, respectively, and $T \subseteq St \times (\mathbf{2}^A \uplus \mathbb{R}_{\geq 0}) \times St$ is the transition relation. ■

To avoid introducing a new definition of timed automata with an interface and multi-action transitions and defining their semantics in terms of ITS, we simply define *projection* for IFTS, which results in an ITS.

5.2.9 Definition (IFTS Projection). The *projection* of an IFTS $S = (St, s_0, A, T, F, fm, \gamma, \Gamma)$ over a set of features Fs is an ITS

$$S \downarrow_{Fs} = (St, s_0, A', T')$$

where A' and T' are defined as

$$A' = \{a \in A \mid FS \models \Gamma(a)\} \qquad T' = \{t \in T \mid FS \models \gamma(t)\}$$

■

Only transitions and actions satisfied by the feature selection Fs are preserved by the projection.

Now we can introduce the definition of strong bisimulation for ITS, and use this notion to defined a product-by-product bisimulation for IFTA.

5.2.10 Definition (Strong Bisimulation for ITS ($\sim$)). Given two ITS S_1 and S_2, over the same set of actions A, S_1 and S_2 are *bisimilar*, denoted $S_1 \sim S_2$, if and only if, there exists a relation $\mathcal{R} \subseteq St_1 \times St_2$, such that $(s_{0_1}, s_{0_2}) \in \mathcal{R}$, and for each $(s_1, s_2) \in \mathcal{R}$,

- $s_1 \xrightarrow{\alpha}_1 s_1'$, $\alpha \in \mathbf{2}^A \uplus \mathbb{R}_{\geq 0}$, then $s_2 \xrightarrow{\alpha}_2 s_2'$ s.t. $(s_1', s_2') \in \mathcal{R}$, and
- $s_2 \xrightarrow{\alpha}_2 s_2'$, $\alpha \in \mathbf{2}^A \uplus \mathbb{R}_{\geq 0}$, then $s_1 \xrightarrow{\alpha}_1 s_1'$ s.t. $(s_1', s_2') \in \mathcal{R}$

■

Intuitively, two IFTS are bisimilar if they share the same feature model, and for each valid combination of features allowed by the feature model, the projected ITS over such combination are bisimilar.

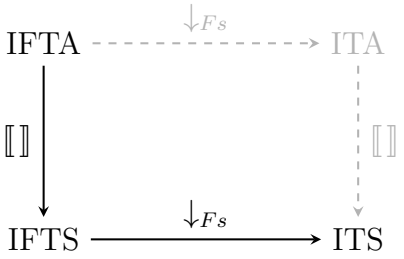
5.2.11 Definition (Strong Bisimulation for IFTS ($\sim$)). Given two IFTS S_1 and S_2, over the same set of actions A and the same feature model fm, S_1 and S_2 are *bisimilar*, denoted $S_1 \sim S_2$, if and only if,

$$\forall\ Fs \in [\![fm]\!]\ \cdot\ S_1 \downarrow_{Fs} \sim\ S_2 \downarrow_{Fs}$$

■

We say two IFTA $\mathcal{A}_1$ and $\mathcal{A}_2$ are bisimilar, $\mathcal{A}_1 \sim \mathcal{A}_2$, if and only if, their underlying transition systems are bisimilar, i.e., if and only if, $[\![\mathcal{A}_1]\!] \sim [\![\mathcal{A}_2]\!]$.

The following figure illustrates the relation between the various formalisms defined here. Gray lines correspond to definitions that have been left out for simplification.



The semantic of an IFTA results in an IFTS (Definition 5.2.3), while the projection of an IFTS over a feature selection *Fs* results in an ITS. If we were to define a kind of Interface TA with multi-action transitions, we could project IFTA into the same *Fs* and obtained an ITA, whose semantics will be an ITS, bisimilar to the one obtained throught the previous step.

Variability-aware

Now we can introduce a variability aware bisimulation relation that takes advantage of the compact structure of IFTS.

5.2.12 Definition (Strong Bisimulation for IFTS ($\sim$)). Given two IFTS S_1 and S_2, over the same set of actions A and the same feature model *fm*, S_1 and S_2 are *bisimilar*, denoted $S_1 \sim S_2$, if and only if, there exists $\mathcal{R} \subseteq St_1 \times St_2$, such that $(s_{0_1}, s_{0_2}) \in \mathcal{R}$, and for each $(s_1, s_2) \in \mathcal{R}$,

- $s_1 \xrightarrow[\varphi_1]{\alpha}_1 s_1'$, $\alpha \in \mathbf{2}^A \uplus \mathbb{R}_{\geq 0}$, then $s_2 \xrightarrow[\varphi_2]{\alpha}_2 s_2'$ s.t. $\models fm \rightarrow (\varphi_1 \rightarrow \varphi_2)$ and $(s_1', s_2') \in \mathcal{R}$, and
- $s_2 \xrightarrow[\varphi_2]{\alpha}_2 s_2'$, $\alpha \in \mathbf{2}^A \uplus \mathbb{R}_{\geq 0}$, then $s_1 \xrightarrow[\varphi_1]{\alpha}_1 s_1'$ s.t. $\models fm \rightarrow (\varphi_2 \rightarrow \varphi_1)$ and $(s_1', s_2') \in \mathcal{R}$

■

The feature expression $fm \rightarrow (\varphi_1 \rightarrow \varphi_2)$ requires that φ_2 should be satisfied in all valid products of the feature model in which φ_1 is satisfied, i.e. whenever $s_1 \xrightarrow{\alpha} s_1'$ is present in a product, so is $s_2 \xrightarrow{\alpha} s_2'$.

A disadvantage of this definition is that it only recognizes two IFTS as bisimilar if they are bisimilar for all the valid set of products specified by their feature model. However, when comparing two systems, it might be of interest to know as well the set of products for which they are bisimilar. We follow Cordy approach [46], and model bisimulation as a function that captures the set of products for which S_1 and S_2 are bisimilar.

5.2.13 Definition (Featured-bisimulation). Given two IFTS S_1 and S_2, over the same sets of actions A, features F, and feature model $fm \in FE(F)$, a featured-bisimulation for S_1 and S_2 is a binary function $\mathcal{R} : St_1 \times St_2 \to FE(F)$ such that

$$\mathcal{R}(s_1, s_2) = \bigwedge_{s_1 \xrightarrow[\varphi_1]{\omega}_1 s_1'} (\varphi_1 \Rightarrow \bigvee_{s_2 \xrightarrow[\varphi_2]{\omega}_2 s_2'} (\varphi_2 \wedge \mathcal{R}(s_1', s_2'))) \wedge$$

$$\wedge \bigwedge_{s_2 \xrightarrow[\varphi_2]{\omega}_2 s_2'} (\varphi_2 \Rightarrow \bigvee_{s_1 \xrightarrow[\varphi_1]{\omega}_1 s_1'} (\varphi_1 \wedge \mathcal{R}(s_1', s_2')))$$

■

Then, let $Fs \in [\![fm]\!]^F$ be a valid product of both IFTS, S_1 and S_2 are bisimilar for Fs if and only if $Fs \models \mathcal{R}(s_{0_1}, s_{0_2})$. Furthermore, let $\mathcal{R}_{max}$ be the largest bisimulation for S_1 and S_2, then $\mathcal{R}_{max}(s_{0_1}, s_{0_2})$ encodes the set of products for which S_1 and S_2 are bisimilar, where by largest it refers to the fact that given any other bisimulation $\mathcal{R}(s_{0_1}, s_{0_2})$, we have that $\mathcal{R}(s_{0_1}, s_{0_2}) \Rightarrow \mathcal{R}_{max}(s_{0_1}, s_{0_2})$. It should be notice that this set of products must be considered in the context of the feature model, this is, it is possible that some feature configuration for which the automata are bisimilar is not allowed by the feature model.

5.2.1 Theorem. *Let S_1 and S_2 be two IFTS over the same set of actions A, features F, and feature model $fm \in FE(F)$. Given a valid feature selection $Fs \in [\![fm]\!]^F$, $Fs \models \mathcal{R}(s_{0_1}, s_{0_2}) \Leftrightarrow S_1 \downarrow_{Fs} \sim S_2 \downarrow_{Fs}$*

Proof. Let us first consider ($\Rightarrow$). If $Fs \models \mathcal{R}(s_{0_1}, s_{0_2})$ we have that for all

$$s_{0_1} \xrightarrow[\varphi_1]{\omega}_1 s_1' \text{ s.t. } Fs \models \varphi_1 \text{ there exists some } s_{0_2} \xrightarrow[\varphi_2]{\omega}_2 s_2' \text{ s.t. } Fs \models \varphi_2 \wedge \mathcal{R}(s_1', s_2')$$

and for all

$$s_{0_2} \xrightarrow[\varphi_2]{\omega}_2 s_2' \text{ s.t. } Fs \models \varphi_2 \text{ there exists some } s_{0_1} \xrightarrow[\varphi_1]{\omega}_1 s_1' \text{ s.t. } Fs \models \varphi_1 \wedge \mathcal{R}(s_1', s_2')$$

In other words, for all transitions $s_{0_1} \xrightarrow[\varphi_1]{\omega}_1 s'_1$ that exist in a product given by Fs, i.e. exist in $S_1 \downarrow_{Fs}$, there is at least a transition $s_{0_2} \xrightarrow[\varphi_2]{\omega}_2 s'_2$ that exist in $S_2 \downarrow_{Fs}$ and such that the same is valid for transitions starting from (s'_1, s'_2); and such that for all $s_{0_2} \xrightarrow[\varphi_2]{\omega}_2 s'_2$ that exist in a product given by Fs, i.e. exist in $S_2 \downarrow_{Fs}$, there is at least a transition $s_{0_1} \xrightarrow[\varphi_1]{\omega}_1 s'_1$ that exist in $S_1 \downarrow_{Fs}$ and such that the same is valid for transitions starting from (s'_1, s'_2). This corresponds exactly with both conditions in Definition 5.2.10. Then, $S_1 \downarrow_{Fs} \sim S_2 \downarrow_{Fs}$.

Let us now consider ($\Leftarrow$). If $S_1 \downarrow_{Fs} \sim S_2 \downarrow_{Fs}$, then let $\mathcal{R}_{Fs}$ be the bisimulation relation for $S_1 \downarrow_{Fs} \sim S_2 \downarrow_{Fs}$ such that $(s_{0_1}, s_{0_2}) \in \mathcal{R}_{Fs}$. By Definition 5.2.10, for all $(s_1, s_2) \in \mathcal{R}_{Fs}$, including (s_{0_1}, s_{0_2}), we have that for all

$$s_1 \xrightarrow{\omega}_{S_1 \downarrow_{Fs}} s'_1 \text{ there exists some } s_2 \xrightarrow{\omega}_{S_2 \downarrow_{Fs}} s'_2 \text{ s.t. } (s'_1, s'_2) \in \mathcal{R}_{Fs}$$

and for all

$$s_2 \xrightarrow{\omega}_{S_2 \downarrow_{Fs}} s'_2 \text{ there exists some } s_1 \xrightarrow{\omega}_{S_1 \downarrow_{Fs}} s'_1 \text{ s.t. } (s'_1, s'_2) \in \mathcal{R}_{Fs}$$

and since these are projections over Fs, we have that

$$\text{if } Fs \models \gamma_{S_1}(s_1 \xrightarrow{\omega}_{S_1} s'_1) \text{ then } Fs \models \gamma_{S_2}(s_2 \xrightarrow{\omega}_{S_2} s'_2), \text{ and}$$
$$\text{if } Fs \models \gamma_{S_2}(s_2 \xrightarrow{\omega}_{S_2} s'_2) \text{ then } Fs \models \gamma_{S_1}(s_1 \xrightarrow{\omega}_{S_1} s'_1)$$

The same is valid for transitions starting from (s'_1, s'_2). Then, $Fs \models \mathcal{R}(s_{0_1}, s_{0_2})$. □

5.2.5 Properties

Operations over IFTA satisfy the usual properties up to *strong bisimulation* ($\sim$).

5.2.2 Theorem. *Given two IFTA $\mathcal{A}_1$ and $\mathcal{A}_2$ with disjoint sets of actions and clocks, we have:*

$$\mathcal{A}_1 \times \mathcal{A}_2 \sim \mathcal{A}_2 \times \mathcal{A}_1 \qquad (\times\text{-commutativity})$$
$$\mathcal{A}_1 \times (\mathcal{A}_2 \times \mathcal{A}_3) \sim (\mathcal{A}_1 \times \mathcal{A}_2) \times \mathcal{A}_3 \qquad (\times\text{-associativity})$$

Proof. Both proofs follow trivially from the definition of product and of the underlying IFTS of each IFTA, and because $\cup$ and $\wedge$ are associative and commutative. In particular, commutativity follows from the fact that

$$\mathcal{R} = \{(\langle (l_1, l_2), \eta \rangle, \langle (l_2, l_1), \eta \rangle) \mid \langle (l_1, l_2), \eta \rangle \in St_{[\![\mathcal{A}_1]\!]}\}$$

is a bisimulation between $\mathcal{A}_1 \times \mathcal{A}_2$ and $\mathcal{A}_2 \times \mathcal{A}_1$. Similarly,

$$\mathcal{R} = \{(\langle (l_1, (l_2, l_3)), \eta \rangle, \langle ((l_1, l_2), l_3), \eta \rangle) \mid \langle (l_1, (l_2, l_3)), \eta \rangle \in St_{[\![\mathcal{A}_1 \times (\mathcal{A}_2 \times \mathcal{A}_3)]\!]}\}$$

is a bisimulation between $\mathcal{A}_1 \times (\mathcal{A}_2 \times \mathcal{A}_3)$ and $(\mathcal{A}_1 \times \mathcal{A}_2) \times \mathcal{A}_3$. □

5.2.3 Theorem. *Given two IFTA $\mathcal{A}_1$ and $\mathcal{A}_2$ with disjoint set of actions and clocks, and actions $a, b, c, d \in A_1$, such that a, b, c, d are different actions, we have:*

$$\Delta_{a,b}\Delta_{c,d}\mathcal{A}_1 \sim \Delta_{c,d}\Delta_{a,b}\mathcal{A}_1 \qquad (\Delta\text{-commutativity})$$
$$(\Delta_{a,b}\mathcal{A}_1) \times \mathcal{A}_2 \sim \Delta_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2) \qquad (\Delta \text{ interacts well with } \times)$$

Proof. Both proof follow trivially by definition of product, synchronization, and of the underlying IFTS of each IFTA. □

5.3 Related Work

Related work is discussed following two lines: compositionality and modularity of SPLs, and compositionality and interfaces for automata.

The importance of having compositional and modular formalisms to model SPLs has been recognized in the literature. In [115], the authors propose an extension to Petri Nets, called Feature Nets (FNs), to incrementally specify the behaviour of an SPL. The approach relies on the design of two types of FNs and their composition: core FNs that model common behaviour to all products in the family, and delta FNs that model variations which can be applied to core behaviour. Another delta-oriented approach is proposed in [108], based on the CCS process calculus, which results in models of features that can be reused easily. They propose as well an incremental approach to verifying SPLs to address scalability issues of the typical product-by-product and family-based approaches. A compositional approach for verification of software product lines has been as well proposed in [111]. The authors propose to model variability requirements at the requirement level and behavioural level. In both cases they use state machines with different levels of abstraction.

There are various papers on interfaces for automata. In [52] de Alfaro *et al.* proposes Interface Automata (IA) to specify temporal aspects of software component interfaces. In this case, temporal does not refers to real time requirements, but the order in which components expect inputs from the environment and the order in which components call external methods (outputs). They propose compatibility checks between interfaces to verify if there exists some environment for a composed system that can make it work; and a notion of refinement for IA. Similarly, in [50], David *et al.* proposes

a specification theory for Timed I/O Automata (TIOA), and defines corresponding notions of composition and refinement for (TIOA), among other operations. However, their theory is based on input enabled automata. In Chapter 6, we propose a notion of refinement for IFTA that takes concepts from both theories. Finally, in [103], Modal I/O automata are proposed as an extension to IA and used to construct a behavioural variability theory for SPL development that can serve to verify if certain requirements can be satisfied from a set of existing assets. However, variability is achieved only by means of must (present in all products) or may (optionally present) transitions.

5.4 Discussion

Interface featured timed automata combines notions from FTA, the theory of automata with interfaces, and component-based design. The result is a formalism capable of modelling SPLs in an incremental and modular way.

In this sense, the main contribution of IFTA is the definition of its composition operation. In addition to making possible incremental design, it allows to verify properties of the composed model against an expected feature model, i.e. to discuss whether the composed model is able to derive the expected products and only those. It is worth mentioning that the approach used to compose feature models is not unique, and there are other approaches that could be explored [1].

The proposed definition has, however, some disadvantages.

First, by forming the product and later applying synchronization, the product automata will possess unnecessary transitions and states that later will be cut, in the case of transitions, or become unreachable, in the case of states. This is inefficient when calculating the composed automata. In practice, it will be better to rename with the same action name interfaces that will be synchronized before computing the product, and define the product to synchronize only transitions with the same shared actions and to interleave only on transitions with non-shared actions. This way, we avoid creating transitions that will later be cut and we remove unreachable states at an early stage. In [37] we document a proof-of-concept prototype to specify, compose, visualize, and translate IFTA models into other formalisms. In this tool we implement both approaches to IFTA composition: synchronization on shared action names, and product followed by synchronization. Section 8.4.2 briefly introduces this prototype.

Second, as discussed in Section 5.2.3, we fix the feature expression of an action before the product in order to achieve commutativity in the composition. This obfuscate the interpretation of feature expression of an action.

In order to simplify the design of synchronous coordination mechanisms, we modelled simultaneous execution of actions as multi-actions associated to transitions. This

is also the approach used by *constraint automata* [17], one of the commonly used formalism to express the semantics of $\mathcal{R}$eo. In practice, there are other ways of achieving this. For example, Uppaal provides committed locations, and non-blocking broadcast. In the latter, an output can synchronized with various inputs in a non-blocking way, i.e., it will synchronize only with the inputs that are enabled at the moment. Non-blocking broadcasting can be achieved through the use of variables and other control mechanism. The disadvantage of these approaches based on committed locations and broadcasting, is to increase complexity in the modelling stage.

Chapter 6

Refinement of Interface Featured Timed Automata

6.1 Introduction

As it happens in the development of any complex system, common and variable assets of an SPL, such as software components, can be designed and developed by different engineers agreeing on a common specification of what their interfaces should be. In this sense, being able to reason about how standalone components, and in this case *families* of components, *implemented* separately satisfy a given *specification* becomes crucial. In this chapter, we propose a notion of *refinement* for real timed software product lines that are modelled as *Interface Featured Timed Automata*.

Refinement allows us to compare two models of the same system presented at different levels of abstraction. The most abstract one is referred to as the *specification*, while the most detailed one is referred to as an *implementation* of the system. If an implementation refines the specification, it agrees with the requirements of the specification in the sense that one may replace the implementation in any context where the specification is used, and still obtain a congruent system. However, since we are dealing with families of components, we need to reason about how a *set of implementations* refine a *set of specifications*.

Figure 6.1 illustrates this problem. The figure shows two composed systems (top): one (top left), composed by an IFTA C, representing a context (here left undefined), and an IFTA P corresponding to the payment selection mechanism in Figure 4.5. The other (top right), which is a refinement of the system on the left, is composed by the same context C, and a new selection mechanism P' (defined in Figure 6.3). The goal is to define a refinement relation that is compositional, i.e. such that it should be sufficient to verify if P' refines P instead of verifying if the composed system on

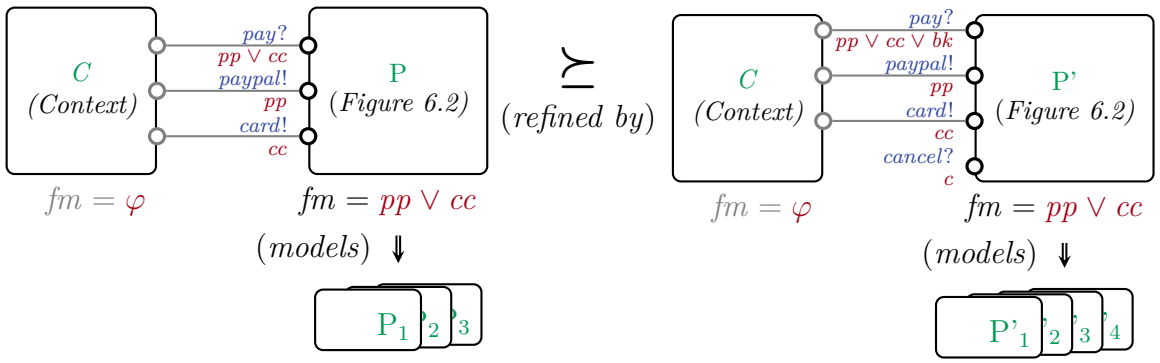


Figure 6.1: Example scenario when reasoning about refinement of families of components. A system composed by two IFTA, C and P (left), is refined by a more detailed system composed by the same IFTA C and a new IFTA P' (right).

the right refines the composed system on the left. However, we need to consider in addition that both automata, P and P', are actually families of components which model different concrete automata, as depicted in Figure 6.1 (bottom). Thus, we need to reason about how each of the new automata P'_i, for $i = 1, \ldots, 4$, refines the automata P_j, for $j = 1, 2$. Informally, the notion of refinement proposed requires that each new automaton of P' refines an automata of P, and that all automata of P are refined by, at least, an automaton of P'. To simplify reasoning, we first define refinement over ITS and then re-build our definition of IFTA refinement over this simpler one.

We show that refinement is a pre-order and congruent with respect to IFTA operations, meaning refinement is compositional. However, as it is discussed in Section 6.2, stronger pre-conditions must be assumed on the variability model of both automata for the congruent result to hold.

In addition, we propose a variability-aware refinement defined directly over IFTS, which takes into account the variability encoded in the model, capturing the set of products for which an IFTS refines another.

Chapter Organization. In Section 6.2 we discuss the product by product refinement relation and its properties. We present the variability-aware refinement in Section 6.3 and conclude by discussing the advantages and limitations of the refinement relation in Section 6.4.

6.2 Refinement

In order to simplify reasoning and allow greater flexibility we separate the notion of refinement into *variability refinement* – which deals with feature model refinement, i.e. with deciding when a set of features can be considered a refinement of another one; and *behavioral refinement* – which captures timed automata refinement. Refinement of timed automata is defined in terms of refinement of the semantic representation of timed transition systems.

There are not many publications in the literature that explore the notion of a refinement relation between two feature models. In [144] the authors propose four kinds of relations between feature models. However, we believe neither of these aligns with a notion of refinement. Informally, we propose that a feature model refines another one if it preserves variability, i.e. if it allows the same set of products, and introduces new variability only in terms of new features.

On the other hand, there exist various notions of automata refinement in the literature, differing on requirements made over the set of actions of the systems being compared (e.g., actions that must be preserved, actions that can be lost), properties inherent to the systems being modelled (e.g., input enabled systems, closed systems, etc.), and properties that the relation should preserve (e.g., safety, reactivity, etc.), among others. The most common relations are simulation and alternating simulation.

Commonly, when dealing with closed systems, i.e., systems that do not interact with the environment, refinement is defined as a simulation relation [16], $T \precsim S$, read S simulates T, where S is the specification and T is the implementation. Intuitively, the implementation can only express behaviour allowed by the specification. The advantage is that simulation preserves all safety properties of the specification. However, when dealing with open systems, as we do in this thesis, simulation is too strict, since it requires the implementation to have the same or less inputs than the specification. On the one hand, this means that a refinement can not incorporate new behaviour in terms of new inputs, which would not be a problem since it would imply no behavioural changes in the resulting system, provided a guarantee that the new inputs are not used. On the other hand, it allows the implementation to have less inputs than the specification. In the case of reactive systems, however, we can not replace a system for another one that reacts to less inputs than the original one. This limits the behaviour of the system, since there will be output actions that are now not captured by the system, but are left unattended.

Thus, when dealing with open systems it is common to define refinement in terms of an *alternating simulation* relation [5, 50, 52, 53]. In this kind of relation, the implementation must simulate all input behaviour of the specification, while the specification must simulate all output behaviour of the implementation. For example, de Alfaro

et. al. [53] introduces Interface Automata, without time, and define the notion of refinement in terms of alternating simulation, extended to support internal steps, i.e., internal actions from both automata which are independent from each other. In [50] David et. al. provided a complete specification theory for Timed I/O Automata where they defined refinement, logical conjunction, structural composition, and a quotient operator. However, their theory is based on input enabled automata.

The rest of this section is divided as follows. We first discuss variability and behavioural refinement and in Section 6.2.1 and Section 6.2.2, respectively. Then we build the definition of IFTA refinement on top of these simpler concepts. Finally, we discuss properties of the refinement relation, and conclude by discussing some advantages and disadvantages of the relation proposed.

6.2.1 Variability Refinement

Thum et al. [144] proposed an algorithm to reason about the relation between two feature models, fm_1 and fm_2, independently of whether they share the exact same set of features, or not. They recognize four type of relations: *refactoring or equivalence* – fm_1 and fm_2 express the same set of products; *specialization* – fm_1 specializes fm_2 if the set of products of fm_1 is a subset of the set of products expressed by fm_2; *generalization* – fm_1 generalizes fm_2 if and only if the set of products of fm_1 is a superset of the set of products of fm_2; and *arbitrary* – otherwise.

However, in order to reason about refinement of families of timed automata we also would like to relate feature models in terms of a refinement relation. There are different ways in which variability refinement can be defined, here we propose one that suits better in order to guarantee congruency with IFTA operations. However, in Section 6.4 we discuss some alternatives that can be considered and briefly hint on what are the main implications of these choices.

Intuitively, a feature model fm_1 refines a feature model fm_2 if when considering the set of features of fm_2, fm_1 expresses exactly the same set of products expressed by fm_2. Thus, fm_1 can add new variability or details only in terms of new features, but must preserve the set of features in fm_2. Formally, if we consider feature models with only terminal features [144], i.e., no abstract features, we define feature model refinement as follows.

6.2.1 Definition (Feature model refinement). Given two feature models $fm_i \in FE(F_i)$ over a set of features F_i, $i = 1, 2$, fm_1 refines fm_2, denoted $fm_1 \sqsubseteq fm_2$, if and

only if,

$$F_1 \supseteq F_2 \qquad \text{(preserves features)}$$
$$[\![fm_1]\!]^{F_1}|_{F_2} = [\![fm_2]\!]^{F_2} \qquad \text{(preserves products)}$$

where $[\![fm]\!]^F|_{F'} = \{FS \cap F' \mid FS \in [\![fm]\!]^F\}$. ■

For example, if consider the payment selection methods P and P' from Figure 6.2, we have that $[\![fm_P]\!] = \{\{pp, cc\}, \{pp\}, \{cc\}\}$ and $[\![fm_{P'}]\!] = \{\{pp, cc, c\}, \{pp, cc\}, \{pp, c\}, \{cc, c\}, \{pp\}, \{cc\}\}$. When we restrict $fm_{P'}$ to consider only features in F_P, we have that $[\![fm_{P'}]\!]|_{F_P} = \{\{pp, cc, \cancel{c}\}, \{pp, cc\}, \{pp, \cancel{c}\}, \{cc, \cancel{c}\}, \{pp\}, \{cc\}\} = [\![fm_P]\!]$, where $\cancel{c}$ means that feature c is removed from the set. Thus, $fm_{P'} \sqsubseteq fm_P$.

However, let us assume that the feature model of P' is now $fm_{P'} = pp \vee cc \vee c$. Then, we have that $[\![fm_{P'}]\!]|_{F_P} = \{\{pp, cc, \cancel{c}\}, \{pp, cc\}, \{pp, \cancel{c}\}, \{cc, \cancel{c}\}, \{pp\}, \{cc\}, \{\cancel{c}\}\} \neq [\![fm_P]\!]$. Because now P' allows a product that does not support neither PayPal nor credit card payments, P' no longer refines P.

6.2.1 Theorem ($\sqsubseteq$ is a partial order). *For any feature model fm_i, for $i = 1, 2, 3$, $fm_1 \sqsubseteq fm_1$; if $fm_1 \sqsubseteq fm_2$ and $fm_2 \sqsubseteq fm_3$, then $fm_1 \sqsubseteq fm_3$; and if $fm_1 \sqsubseteq fm_2$ and $fm_2 \sqsubseteq fm_1$, then $fm_1 \equiv fm_2$.*

Proof. The reflexive and antisymmetric properties are trivial by definition of set inclusion and set intersection.

In the case of transitivity, first note that by transitivity of set inclusion we have that $F_1 \supseteq F_3$. Then,

$$\bigcup_{Fs \in [\![fm_1]\!]^{F_1}} Fs \cap F_2 = [\![fm_2]\!]^{F_2} \qquad \text{by } fm_1 \sqsubseteq fm_2 \tag{6.1}$$
$$\bigcup_{Fs \in [\![fm_2]\!]^{F_2}} Fs \cap F_3 = [\![fm_3]\!]^{F_3} \qquad \text{by } fm_2 \sqsubseteq fm_3 \tag{6.2}$$
$$\bigcup_{Fs \in [\![fm_1]\!]^{F_1}} Fs \cap F_2 \cap F_3 = [\![fm_3]\!]^{F_3} \qquad \text{by (6.1) and (6.2)} \tag{6.3}$$
$$\bigcup_{Fs \in [\![fm_1]\!]^{F_1}} Fs \cap F_3 = [\![fm_3]\!]^{F_3} \qquad \text{by (6.3) and } F_2 \supseteq F_3 \tag{6.4}$$

where $\bigcup_{Fs \in [\![fm_1]\!]^{F_1}} (Fs \cap F_3)$ is exactly the definition of $[\![fm_1]\!]^{F_1}|_{F_3}$.

Thus, $fm_1 \sqsubseteq fm_3$, and $\sqsubseteq$ is a partial order. □

6.2.2 Behavioural Refinement

Refinement allows to verify if a given *implementation* agrees with a *specification*. We consider implementations as automata that are more detailed specifications. Intuitively, an automata $\mathcal{A}$ that refines an automata $\mathcal{B}$ should be able to replace $\mathcal{B}$ in every context in which $\mathcal{B}$ appears. Our notion of refinement is similar to the one in [52], where there is an alternating simulation between both automata: $\mathcal{A}$ must simulate all input behavior of $\mathcal{B}$, while $\mathcal{B}$ must simulate all output behavior from $\mathcal{A}$. Thus, $\mathcal{A}$ can allow more legal inputs, and fewer outputs, than $\mathcal{B}$.

Similarly to [50] we define refinement at the semantic level, i.e., at the level of transition systems and then build up towards IFTA refinement. Because we separate refinement into variability and behavioural refinement, our base notion of refinement is defined for Interface Transition Systems (Definition 5.2.8), i.e., IFTS without variability.

In fact, our notion of refinement can be seen as an extension of [52] for timed systems with multi-action transitions. Here as well, the definition of refinement must consider the fact that both automata have internal actions which are independent from each other. Additionally, since we are dealing with timed transition systems, the definition of refinement must consider that internal steps can incorporate delays. Thus, we define a transition relation that captures all transition steps possible from a state s to a state s' by any combination of internal and delay steps.

6.2.2 Definition. Given an ITS S and states $s, s' \in St_S$, notation $s \overset{\tau^*}{\Longrightarrow}{}^{d}_{S} s'$ means that there is a sequence of transition steps from T_S, such that

$$s \xrightarrow[S]{\omega_i} s_1 \dots s_n \xrightarrow[S]{\omega_n} s'$$

with $\omega_i \in \tau \uplus \mathbb{R}_{\geq 0}$ and such that $(\sum_{\omega_i \in \mathbb{R}_{\geq 0}} \omega_i) = d$. For simplicity, we use $s \overset{\omega}{\Longrightarrow}{}^{d}_{S} s'$ if there is a sequence of transition steps from T_S, such that

$$s \overset{\tau^*}{\Longrightarrow}{}^{d}_{S} s_n \xrightarrow[S]{\omega} s'$$

with $\omega \in 2^A$. ■

In this context, ITS refinement is defined as follows.

6.2.3 Definition (Refinement of ITS). Given two ITS, S and T, such that $I_T \subseteq I_S$ and $O_S \subseteq O_T$, S refines T, denoted $S \preceq T$, if and only if, there exists a relation $\mathcal{R} \subseteq St_S \times St_T$, such that $(s_0, t_0) \in \mathcal{R}$ and for each $(s, t) \in \mathcal{R}$,

1. $s \stackrel{\tau^*}{\Longrightarrow}{}^d_S\ s', d \in \mathbb{R}_{\geq 0}$ then $t \stackrel{\tau^*}{\Longrightarrow}{}^d_T\ t'$ *and* $(s', t') \in \mathcal{R}$, *for some* $t' \in St_T$

2. $s \stackrel{OIs}{\Longrightarrow}{}^d_S\ s', d \in \mathbb{R}_{\geq 0}, O \neq \emptyset$ then $t \stackrel{OIs}{\Longrightarrow}{}^d_T\ t'$ and $(s', t') \in \mathcal{R}$ for some $t' \in St_T$

3. $t \stackrel{IO}{\Longrightarrow}{}^d_T\ t', d \in \mathbb{R}_{\geq 0}, I \neq \emptyset$ then $s \stackrel{IO}{\Longrightarrow}{}^d_S\ s'$ and $(s', t') \in \mathcal{R}$ for some $s' \in St_S$

where I_s is either $\emptyset$, or has only inputs shared by both automata, $I_s \subseteq I_T$. ■

Condition 1 expresses that any delay allowed from s, must be a delay allowed from t, possible by taken some internal steps. Condition 2 expresses that any output transition, which may have inputs that are *shared* by both systems, and that can be taken from s after a delay d, possible by taking some internal steps, must simulate a (sequence of) transition(s) from t. Notice that if there is a multi-action transition with outputs and inputs, such that the inputs include inputs available only in the implementation, this is considered as new behaviour incorporated by the use of new inputs, and as such, it is ignored. Condition 3 expresses that any input transition, with possible outputs, taken from t after a delay d, possibly with some internal steps, must be simulated by a (sequence of) transition(s) from s.

In comparison with de Alfaro et. al., we relax some of the requirements made over the states being compared, s and t. In particular, when considering input labeled transitions (condition 3), de Alfaro et. al. defines that s and t are in a refinement relation, only if, whenever in t is possible to receive an input, s may receive the same input. Here, we require that whenever in t is possible to receive an input within a certain time, possibly trough a sequence of internal steps, s may receive the same input within the same time, possibly through a series of internal steps. In other words, de Alfaro requires T and S to be immediately ready to receive the input in state t and s, respectively, while we required that S must receive the input within the same amount of time that T from t, but may still perform internal actions from s before that.

6.2.3 IFTA Refinement

Before formalizing refinement for families of timed automata, let us consider refinement for IFTS. Informally, given two IFTS, S and T, S refines T if for each valid product in S, the projection of S onto such a product refines the projection of T onto the same product. However, depending on the relation existing between the set of products of S and T, this can lead to different notions of refinement. As presented in Section 6.2.1, we propose that S should preserve the variability of T, i.e. S should allow exactly the same products as T, although it may also increase the set of features and allow

more products with respect to the new features. We will discuss other possibilities in Section 6.4. Formally, refinement of IFTS and IFTA are defined as follows.

6.2.4 Definition (Refinement of IFTS). Given two IFTS, S and T, S refines T, denoted $S \preceq T$, if and only if,

$$fm_S \sqsubseteq fm_T \qquad \text{(variability refinement)}$$
$$\forall\ FS \in [\![fm_S]\!]^{Fs}\ \cdot\ S\downarrow_{Fs} \preceq\ T\downarrow_{Fs} \qquad \text{(behaviour refinement)}$$

■

6.2.5 Definition (Refinement of IFTA). Given two grounded IFTA $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A}$ refines $\mathcal{B}$, denoted $\mathcal{A} \preceq \mathcal{B}$, if and only if, $[\![\mathcal{A}]\!] \preceq [\![\mathcal{B}]\!]$. ■

Figure 6.2 shows an implementation of a family of payment selection methods, P' (right), which refines the IFTA P (left), previously introduced in Figure 5.3. The new automaton introduces a new input, *cancel* that depends on a new feature c which represents the support for cancelling the payment request. In addition, P' ensures that the method of payment or the cancellation will be chosen faster than in P, as indicated by the invariant $c \leq 3$.

Figure 6.3 shows a more complex example of refinement incorporating internal actions (represented with a ; after the action name). The IFTA represents a more detailed implementation of the automaton PP introduced in Figure 5.4. The specification requires that whenever the user makes a payment through PayPal, the system will issue an error or a success signal in less than ten units of time. The implementation deals with the actual login into PayPal and confirmation of the payment. In PP', after the user requests to issue a payment through PayPal, the user must login within 5 units of time, or an error will be issued. The log in can be successful or can issue an error in less than one unit of time. In case the user login is successful, a confirmation of the payment must be issued in less than one unit of time after which the system issues a signal of error or success. Both, PP and PP', share the same feature model. The implementation PP' guaranties that whenever a payment is made through PayPal, the system will issue an error or success signal in less than seven units of time, satisfying the requirements of PP. Thus, $PP' \preceq PP$.

6.2.4 Properties

Refinement of IFTA is a pre-order and it is compositional. The latter allows decomposition of refinement proofs, improving efficiency in refinement checking.

6.2.2 Theorem ($\preceq$ is pre-order). *For any grounded IFTA $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, $A_1 \preceq A_1$, and if $\mathcal{A}_1 \preceq \mathcal{A}_2$ and $\mathcal{A}_2 \preceq \mathcal{A}_3$, then $\mathcal{A}_1 \preceq \mathcal{A}_3$.*

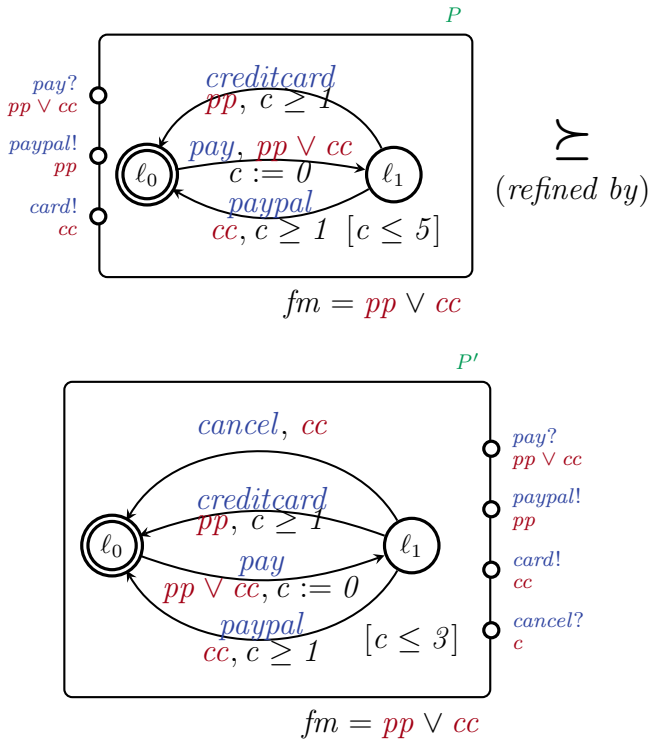


Figure 6.2: Example of a family of payment selection methods P' with new variability, interfaces and time restrictions, refining the family P.

Proof. $\mathcal{A}_1 \preceq \mathcal{A}_1$ is trivial by definition of $\preceq$. For transitivity note that by transitivity of $\sqsubseteq$ we have $fm_1 \sqsubseteq fm_3$. Then, by definition of $\preceq$ and transitivity of $\sqsubseteq$, we have that

$$\forall_{\ Fs \in [\![fm_1]\!]^{F_1}} \ \cdot\ [\![A_2]\!]\downarrow_{Fs} \preceq [\![A_3]\!]\downarrow_{Fs} \tag{6.5}$$

This is, projecting a transition system into a set of features containing a valid feature selection and features that are not part of the system, results in the same projection than projecting only against the valid set of features.

Then, let us assume that $\forall_{\ Fs \in [\![fm_1]\!]^{F_1}}$,

$$\mathcal{R}^{12}_{Fs} \subseteq St_1 \times St_2, \ \textit{with}\ (s_{0_1}, s_{0_2}) \in \mathcal{R}^{12}_{Fs}$$
$$\mathcal{R}^{23}_{Fs} \subseteq St_2 \times St_3, \ \textit{with}\ (s_{0_2}, s_{0_3}) \in \mathcal{R}^{23}_{Fs}$$

are the refinement relations for $A_1 \preceq A_2$, and $A_2 \preceq A_3$, respectively.

For all $Fs \in [\![fm_1]\!]^{F_1}$, we show there exists a relation $\mathcal{R}^{13}_{Fs}$, such that $(s_{0_1}, s_{0_3}) \in \mathcal{R}^{13}_{Fs}$, and each $(s_1, s_3) \in \mathcal{R}^{13}_{Fs}$ satisfies the conditions of Definition 6.2.3 as follows.

$\forall_{\ Fs \in [\![fm_1]\!]^{F_1}}$, we have that for each $(s_1, s_2) \in \mathcal{R}^{12}_{Fs}$, including (s_{0_1}, s_{0_2})

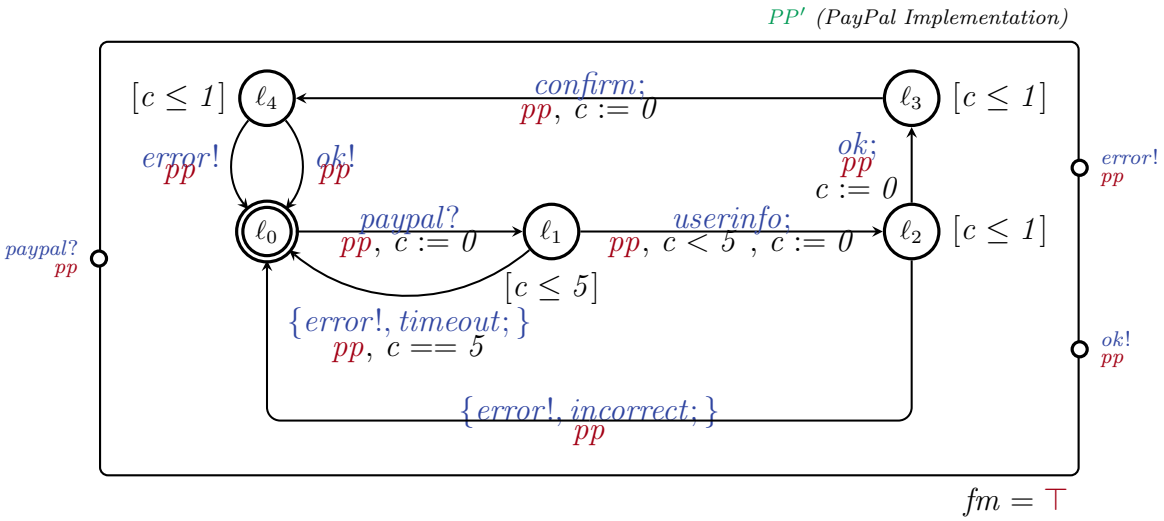


Figure 6.3: An example of IFTA refinement with internal actions, where PP' refines the IFTA PP from Figure 5.4

.

(delay) if there exists some $s_1 \overset{\tau^*}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_1]\!]\downarrow_{Fs}} s_1'$ then by $\mathcal{A}_1 \preceq \mathcal{A}_2$ there exists some $s_2 \overset{\tau^*}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_2]\!]\downarrow_{Fs}} s_2'$ such that $(s_1', s_2') \in \mathcal{R}^{12}_{Fs}$, and, by $\mathcal{A}_2 \preceq \mathcal{A}_3$ and (6.5), we know that there exists some $(s_2, s_3) \in R^{23}_{Fs}$ such that $s_3 \overset{\tau^*}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_3]\!]\downarrow_{Fs}} s_3'$ and $(s_2', s_3') \in R^{23}_{Fs}$. Thus, $\{(s_1, s_3), (s_1', s_3')\} \subseteq R^{13}_{Fs}$.

(output) if there exists some $s_1 \overset{OI_s}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_1]\!]\downarrow_{Fs}} s_1'$, for $O \neq \emptyset$ and $I_s = \emptyset$ or $I_s \subseteq I_3$, then, by $\mathcal{A}_1 \preceq \mathcal{A}_2$, there exists some $s_2 \overset{OI_s}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_2]\!]\downarrow_{Fs}} s_2'$ such that $(s_1', s_2') \in \mathcal{R}^{12}_{Fs}$, and, by $\mathcal{A}_2 \preceq \mathcal{A}_3$ and (6.5), we know that there exists some $(s_2, s_3) \in R^{23}_{Fs}$ such that $s_3 \overset{OI_s}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_3]\!]\downarrow_{Fs}} s_3'$ and $(s_2', s_3') \in R^{23}_{Fs}$. Thus, $\{(s_1, s_3), (s_1', s_3')\} \subseteq R^{13}_{Fs}$.

(input) if there exists some $s_3 \overset{IO}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_3]\!]\downarrow_{Fs}} s_3'$ for $I \neq \emptyset$, then there exists some $s_2 \overset{IO}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_2]\!]\downarrow_{Fs}} s_2'$, and, by $\mathcal{A}_1 \preceq \mathcal{A}_2$ and (6.5), we know that there exists some $(s_1, s_2) \in R^{12}_{Fs}$, such that $s_1 \overset{IO}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_1]\!]\downarrow_{Fs}} s_1'$ and $(s_1', s_2') \in R^{12}_{Fs}$. Thus, $\{(s_1, s_3), (s_1', s_3')\} \subseteq R^{13}_{Fs}$.

Therefore $\mathcal{A}_1 \preceq \mathcal{A}_3$, and refinement is a pre-order. □

In order to be compositional, refinement must be congruent with respect to IFTA operations, i.e., product and synchronization. However, stronger pre-conditions are required to ensure congruency for both operations.

In both cases the problem arises with feature model refinement. In the case of product, let us consider three automata $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{B}$ such that $\mathcal{A}_1 \preceq \mathcal{A}_2$. We want to ensure that $\mathcal{A}_1 \times \mathcal{B} \preceq \mathcal{A}_2 \times \mathcal{B}$ holds. The problem is that $\mathcal{A}_1$ and $\mathcal{B}$ can share features as well as $\mathcal{A}_2$ and $\mathcal{B}$. Thus, we can not infer how each product will affect the resulting feature model. As an example, consider the following arbitrary feature models,

$$\begin{aligned}[\![fm_1]\!]^{F_1} &= \{\{f_1, f_2, f_3\}, \{f_1, f_3\}, \{f_3\}\} \\ [\![fm_2]\!]^{F_2} &= \{\{f_1, f_2\}, \{f_1\}, \{\}\} \\ [\![fm_B]\!]^{F_B} &= \{\{f_1, f_2, f_3\}, \{f_4\}\}\end{aligned}$$

with $F_1 = \{f_1, f_2, f_3\}$, $F_2 = \{f_1, f_2\}$, and $F_B = \{f_1, f_2, f_3, f_4\}$. We have that $[\![fm_1]\!]^{F_1}|_{F_2} = [\![fm_2]\!]^{F_2}$, thus $fm_1 \sqsubseteq fm_2$. However, when we do the corresponding feature model conjunctions (as in the product), we have that $[\![fm_1 \wedge fm_B]\!]^{F_{1B}} = \{\{f_1, f_2, f_3\}\}$ and $[\![fm_2 \wedge fm_B]\!]^{F_{2B}} = \{\{f_1, f_2, f_3\}, \{f_4\}\}$. Thus, we have that

$$[\![fm_1 \wedge fm_B]\!]^{F_{1B}}|_{F_2 \cup F_B} = \{\{f_1, f_2, f_3\}\} \neq [\![fm_2 \wedge fm_B]\!]^{F_{2B}}$$

and $fm_{1B} \not\sqsubseteq fm_{2B}$.

In order to guarantee that refinement is congruent with respect to product we fix the assumption that neither $\mathcal{A}_1$ and $\mathcal{B}$, nor $\mathcal{A}_2$ and $\mathcal{B}$ share any features.

6.2.3 Theorem ($\preceq$ congruence w.r.t. $\times$). *For any grounded IFTA $\mathcal{A}_1$, $\mathcal{A}_2$, and B, such that $\mathcal{A}_i$ and B have disjoint set of features, for $i = 1, 2$, if $\mathcal{A}_1 \preceq \mathcal{A}_2$, then $\mathcal{A}_1 \times B \preceq \mathcal{A}_2 \times B$.*

Proof. Let us consider first feature model refinement. First, by $fm_1 \sqsubseteq fm_2$ we have that $F_1 \supseteq F_2$, and therefore $F_1 \cup F_{\mathcal{B}} \supseteq F_2 \cup F_{\mathcal{B}}$. Now, we want to show that $[\![fm_1 \wedge fm_{\mathcal{B}}]\!]^{F_1 \cup F_{\mathcal{B}}}|_{F_2 \cup F_{\mathcal{B}}} = [\![fm_2 \wedge fm_{\mathcal{B}}]\!]^{F_2 \cup F_{\mathcal{B}}}$. According to the semantics of a feature expression, and because $F_1 \cap F_{\mathcal{B}} = F_2 \cap F_{\mathcal{B}} = \emptyset$, we have the following equivalences.

$$\begin{aligned}[\![fm_1 \wedge fm_{\mathcal{B}}]\!]^{F_1 \cup F_{\mathcal{B}}}|_{F_2 \cup F_{\mathcal{B}}} = \{(Fs_1 \cap F_2) \cup Fs_{\mathcal{B}} \mid\ & Fs_1 \subseteq F_1,\ Fs_{\mathcal{B}} \subseteq F_{\mathcal{B}}, \\ & Fs_1 \models fm_1 \text{ and} \\ & Fs_{\mathcal{B}} \models F_{\mathcal{B}}\}\end{aligned} \tag{6.6}$$

$$\begin{aligned}[\![fm_2 \wedge fm_{\mathcal{B}}]\!]^{F_2 \cup F_{\mathcal{B}}} = \{Fs_2 \cup Fs_{\mathcal{B}} \mid\ & Fs_2 \subseteq F_2,\ Fs_{\mathcal{B}} \subseteq F_{\mathcal{B}}, \\ & Fs_2 \models fm_2 \text{ and } Fs_{\mathcal{B}} \models F_{\mathcal{B}}\}\end{aligned} \tag{6.7}$$

By $fm_1 \sqsubseteq fm_2$ we know that every $Fs_1 \cap F_2$ such that $Fs_1 \subseteq F_1$ and $Fs_1 \models fm_1$ are exactly all the sets $Fs_2 \subseteq F_2$ such that $Fs_2 \models fm_2$. Therefore, equation (6.6) is equivalent to (6.7), and $fm_{1\mathcal{B}} \sqsubseteq fm_{2\mathcal{B}}$.

Now, let us consider behavioral refinement. Assume that $\forall\ Fs \in [\![fm_1]\!]^{F_1}$,

$$\mathcal{R}^{12}_{Fs} \subseteq St_1 \times St_2, \ \textit{with}\ (s_{0_1}, s_{0_2}) \in \mathcal{R}^{12}_{Fs}$$

is the refinement relation for $A_1 \preceq A_2$.

We want to show that, for all $Fs \in [\![fm_{1\mathcal{B}}]\!]^{F_{1\mathcal{B}}}$, there exists a relation $\mathcal{R}^{1\mathcal{B}2\mathcal{B}}_{Fs}$, such that $(s_{0_{1b}}, s_{0_{2b}}) \in \mathcal{R}^{1\mathcal{B}2\mathcal{B}}_{Fs}$, and each $(s_{1b}, s_{2b}) \in \mathcal{R}^{1\mathcal{B}2\mathcal{B}}_{Fs}$ satisfies the conditions in Definition 6.2.3.

For any $Fs \in [\![fm_{1\mathcal{B}}]\!]^{F_{1\mathcal{B}}}$, we have that:

(delay) if there exists some $\langle (l_1, l_b), \eta^{\mathcal{A}_1\mathcal{B}} \rangle \overset{\tau^*}{\Longrightarrow}^{d}_{[\![\mathcal{A}_1 \times \mathcal{B}]\!]\downarrow_{Fs}} \langle (l'_1, l'_b), \eta^{\mathcal{A}'_1\mathcal{B}'} \rangle$, by definition of IFTA product, there exists some $\langle l_1, \eta^{\mathcal{A}_1} \rangle \overset{\tau^*}{\Longrightarrow}^{d}_{[\![\mathcal{A}_1]\!]\downarrow_{Fs}} \langle l'_1, \eta^{\mathcal{A}'_1} \rangle$ and there exists some $\langle l_b, \eta^{\mathcal{B}} \rangle \overset{\tau^*}{\Longrightarrow}^{d}_{[\![\mathcal{B}]\!]\downarrow_{Fs}} \langle l'_b, \eta^{\mathcal{B}'} \rangle$; by $\mathcal{A}_1 \preceq \mathcal{A}_2$, we have that there exists some $\langle l_2, \eta^{\mathcal{A}_2} \rangle \overset{\tau^*}{\Longrightarrow}^{d}_{[\![\mathcal{A}_2]\!]\downarrow_{Fs}} \langle l'_2, \eta^{\mathcal{A}'_2} \rangle$, such that $(\langle l_1, \eta^{\mathcal{A}_1} \rangle, \langle l_2, \eta^{\mathcal{A}_2} \rangle) \in \mathcal{R}^{12}_{Fs\setminus F_{\mathcal{B}}}$ [1] and $(\langle l'_1, \eta^{\mathcal{A}'_1} \rangle, \langle l'_2, \eta^{\mathcal{A}'_2} \rangle) \in \mathcal{R}^{12}_{FS\setminus F_{\mathcal{B}}}$. Thus, by definition of IFTA product, there exists some $\langle (l_2, l_b), \eta^{\mathcal{A}_2\mathcal{B}} \rangle \overset{\tau^*}{\Longrightarrow}^{d}_{[\![\mathcal{A}_2 \times \mathcal{B}]\!]\downarrow_{Fs}} \langle (l'_2, l'_b), \eta^{\mathcal{A}'_2\mathcal{B}'} \rangle$, such that $(\langle (l_1, l_b), \eta^{\mathcal{A}_1\mathcal{B}} \rangle, \langle (l_2, l_b), \eta^{\mathcal{A}_2\mathcal{B}} \rangle) \in \mathcal{R}^{1\mathcal{B}2\mathcal{B}}_{Fs}$, and $(\langle (l'_1, l'_b), \eta^{\mathcal{A}'_1\mathcal{B}'} \rangle, \langle (l'_2, l'_b), \eta^{\mathcal{A}'_2\mathcal{B}'} \rangle) \in \mathcal{R}^{1\mathcal{B}2\mathcal{B}}_{Fs}$.

(output) if there exists some $\langle (l_1, l_b), \eta^{\mathcal{A}_1\mathcal{B}} \rangle \overset{OI}{\Longrightarrow}^{d}_{[\![\mathcal{A}_1 \times \mathcal{B}]\!]\downarrow_{Fs}} \langle (l'_1, l'_b), \eta^{\mathcal{A}'_1\mathcal{B}'} \rangle$, we have to consider three cases depending on which automaton takes the observable transition: 1) *independent transition from* $\mathcal{A}_1$, i.e., $OI \subseteq O_1 \cup I_2$; 2) *independent transition from* $\mathcal{B}$, i.e., $OI \subseteq O_{\mathcal{B}} \cup I_{\mathcal{B}}$; or 3) *simultaneous transition from* $\mathcal{A}_1$ and $\mathcal{B}$, i.e., $OI \subseteq O_{1\mathcal{B}} \cup I_{2\mathcal{B}}$. Here we consider case 1) only; the reasoning is analogous for the other cases. In this case, by definition of IFTA product, we have that there exists some $\langle l_1, \eta^{\mathcal{A}_1} \rangle \overset{OI}{\Longrightarrow}\ ^{d}_{[\![\mathcal{A}_1]\!]\downarrow_{Fs}} \langle l'_1, \eta^{\mathcal{A}'_1} \rangle$ and there exists some $\langle l_b, \eta^{\mathcal{B}} \rangle \overset{\tau^*}{\Longrightarrow}^{d}_{[\![\mathcal{B}]\!]\downarrow_{Fs}} \langle l'_b, \eta^{\mathcal{B}'} \rangle$. Then, by $\mathcal{A}_1 \preceq \mathcal{A}_2$, we have that there exists some $\langle l_2, \eta^{\mathcal{A}_2} \rangle \overset{OI}{\Longrightarrow}\ ^{d}_{[\![\mathcal{A}_2]\!]\downarrow_{Fs}} \langle l'_2, \eta^{\mathcal{A}_2} \rangle$ such that $(\langle l_1, \eta^{\mathcal{A}_1} \rangle, \langle l_2, \eta^{\mathcal{A}_2} \rangle) \in \mathcal{R}^{12}_{FS\setminus F_{\mathcal{B}}}$ and $(\langle l'_1, \eta^{\mathcal{A}'_1} \rangle, \langle l'_2, \eta^{\mathcal{A}'_2} \rangle) \in \mathcal{R}^{12}_{FS\setminus F_{\mathcal{B}}}$. Thus, by definition of IFTA product, we have

[1] Note that $Fs \in [\![fm_{1\mathcal{B}}]\!]$, and that $Fs \setminus F_B \in [\![fm_1]\!]$ since $F_1 \cap F_B = \emptyset$. Thus, we use $[\![\mathcal{A}_1]\!]\downarrow_{Fs}$ instead of $[\![\mathcal{A}_1]\!]\downarrow_{Fs\setminus F_B}$ since this results in the same automaton. Similarly, we use $[\![\mathcal{A}_2]\!]\downarrow_{Fs}$ instead of $[\![\mathcal{A}_2]\!]\downarrow_{Fs\setminus F_B}$.

that there exists some $\langle (l_2, l_b), \eta^{\mathcal{A}_2\mathcal{B}} \rangle \overset{OI}{\Longrightarrow} \overset{d}{[\![\mathcal{A}_2 \times \mathcal{B}]\!]_{\downarrow Fs}} \langle (l'_2, l'_b), \eta^{\mathcal{A}'_2\mathcal{B}'} \rangle$, such that $(\langle (l_1, l_b), \eta^{\mathcal{A}_1\mathcal{B}} \rangle, \langle (l_2, l_b), \eta^{\mathcal{A}_2\mathcal{B}} \rangle) \in \mathcal{R}^{1\mathcal{B}2\mathcal{B}}_{Fs}$ and $(\langle (l'_1, l'_b), \eta^{\mathcal{A}'_1\mathcal{B}'} \rangle, \langle (l'_2, l'_b), \eta^{\mathcal{A}'_2\mathcal{B}'} \rangle) \in \mathcal{R}^{1\mathcal{B}2\mathcal{B}}_{Fs}$.

(input) if there exists some $\langle (l_2, l_b), \eta^{\mathcal{A}_2\mathcal{B}} \rangle \overset{IO}{\Longrightarrow} \overset{d}{[\![\mathcal{A}_2 \times \mathcal{B}]\!]_{\downarrow Fs}} \langle (l'_2, l'_b), \eta^{\mathcal{A}'_2\mathcal{B}'} \rangle$, similarly as before, we have to consider three cases depending on which automaton takes the observable transition: 1) *independent transition from* $\mathcal{A}_2$, i.e., $IO \subseteq I_2 \cup O_2$; 2) *independent transition from* $\mathcal{B}$, i.e., $IO \subseteq I_\mathcal{B} \cup O_\mathcal{B}$; or 3) *simultaneous transition from* $\mathcal{A}_2$ and $\mathcal{B}$, i.e., $IO \subseteq I_{2\mathcal{B}} \cup O_{2\mathcal{B}}$. As before, we consider only case 1); the reasoning is analogous for the other cases. In this case, by definition of IFTA product, we have that there exists some $\langle l_2, \eta^{\mathcal{A}_2} \rangle \overset{IO}{\Longrightarrow} \overset{d}{[\![\mathcal{A}_2]\!]_{\downarrow Fs}} \langle l'_2, \eta^{\mathcal{A}'_2} \rangle$ and there exists some $\langle l_b, \eta^{\mathcal{B}} \rangle \overset{\tau^*}{\Longrightarrow} \overset{d}{[\![\mathcal{B}]\!]_{\downarrow Fs}} \langle l'_b, \eta^{\mathcal{B}'} \rangle$. Then, by $\mathcal{A}_1 \preceq \mathcal{A}_2$, we have that there exists some $\langle l_1, \eta^{\mathcal{A}_1} \rangle \overset{IO}{\Longrightarrow} \overset{d}{[\![\mathcal{A}_1]\!]_{\downarrow Fs}} \langle l'_1, \eta^{\mathcal{A}_1} \rangle$, such that $(\langle l_1, \eta^{\mathcal{A}_1} \rangle, \langle l_2, \eta^{\mathcal{A}_2} \rangle) \in \mathcal{R}^{12}_{FS \setminus F_\mathcal{B}}$ and $(\langle l'_1, \eta^{\mathcal{A}'_1} \rangle, \langle l'_2, \eta^{\mathcal{A}'_2} \rangle) \in \mathcal{R}^{12}_{FS \setminus F_\mathcal{B}}$. Thus, by definition of IFTA product, we have that there exists some $\langle (l_1, l_b), \eta^{\mathcal{A}_1\mathcal{B}} \rangle \overset{IO}{\Longrightarrow} \overset{d}{[\![\mathcal{A}_1 \times \mathcal{B}]\!]_{\downarrow Fs}} \langle (l'_1, l'_b), \eta^{\mathcal{A}'_1\mathcal{B}'} \rangle$, such that $(\langle (l_1, l_b), \eta^{\mathcal{A}_1\mathcal{B}} \rangle, \langle (l_2, l_b), \eta^{\mathcal{A}_2\mathcal{B}} \rangle) \in \mathcal{R}^{1\mathcal{B}2\mathcal{B}}_{Fs}$ and $(\langle (l'_1, l'_b), \eta^{\mathcal{A}'_1\mathcal{B}'} \rangle, \langle (l'_2, l'_b), \eta^{\mathcal{A}'_2\mathcal{B}'} \rangle) \in \mathcal{R}^{1\mathcal{B}2\mathcal{B}}_{Fs}$.

Thus, we have that $\mathcal{A}_1 \times \mathcal{B} \preceq \mathcal{A}_2 \times \mathcal{B}$. □

In the case of synchronization, a similar problem arises as well with feature model refinement. Intuitively, by definition of refinement, in the implementation an input can be present in more products, and an output can be present in less products than in the specification. Thus, it is natural that the feature expressions associated to the input and output that we want to synchronize in the implementation differ from the feature expressions in the specification. Thus, if an implementation refines a specification, after synchronization, the feature model of the implementation does not necessarily refines the feature model of the specification.

Intuitively, a possible solution is to require that an implementation can only replace the specification if it does not add new connections and maintains all connections already in the specification. This means that, for each valid product in the implementation, the corresponding automata in the implementation can be synchronized over a given set of input and outputs, if and only if, the corresponding automata in the specification can be synchronized over the same inputs and outputs. The following theorem captures this property.

6.2.4 Theorem ($\preceq$ congruence w.r.t. Δ). *For any grounded IFTA $\mathcal{A}_1$, $\mathcal{A}_2$, and actions i, o such that $(i, o) \in I_j \times O_j$ for $j = 1, 2$, if $\mathcal{A}_1 \preceq \mathcal{A}_2$, then $\Delta_{i,o}\mathcal{A}_1 \preceq \Delta_{i,o}\mathcal{A}_2$, only if, $fm_1 \rightarrow ((\Gamma_1(i) \leftrightarrow \Gamma_1(o)) \leftrightarrow (\Gamma_2(i) \leftrightarrow \Gamma_2(o)))$ is satisfiable.*

Proof. Let us consider feature model refinement. First, by $fm_1 \sqsubseteq fm_2$ and because Δ does not affect features, we have that $F_{\Delta 1} = F_1 \supseteq F_2 = F_{\Delta 2}$. Second, we want to show that $[\![fm_1 \wedge (\Gamma_1(i) \leftrightarrow \Gamma_1(o))]\!]^{F_{\Delta 1}}|_{F_{\Delta 2}} = [\![fm_2 \wedge (\Gamma_2(i) \leftrightarrow \Gamma_2(o))]\!]^{F_{\Delta 2}}$. According to the semantics of a feature expression, we have the following equivalences:

$$[\![fm_1 \wedge (\Gamma_1(i) \leftrightarrow \Gamma_1(o))]\!]^{F_{\Delta 1}}|_{F_{\Delta 2}} = \{FS_1 \cap F_{\Delta 2} \mid FS_1 \subseteq F_{\Delta 1}, \; FS_1 \models fm_1, \; and \; FS_1 \models (\Gamma_1(i) \leftrightarrow \Gamma_1(o))\} \tag{6.8}$$

$$[\![fm_2 \wedge (\Gamma_2(i) \leftrightarrow \Gamma_2(o))]\!]^{F_{\Delta 2}} = \{FS_2 \mid FS_2 \subseteq F_{\Delta 2}, \; FS_2 \models fm_2, \; and \; FS_2 \models (\Gamma_2(i) \leftrightarrow \Gamma_2(o))\} \tag{6.9}$$

By $fm_1 \sqsubseteq fm_2$ we know that all sets $FS_1 \cap F_{\Delta 2}$ such that $FS_1 \subseteq F_{\Delta 1}$ and $FS_1 \models fm_1$, where $F_{\Delta 1} = F_1$, are exactly all the sets $FS_2 \subseteq F_{\Delta 2}$ such that $FS_2 \models fm_2$. In addition, we know that $FS_1 \models \Gamma_1(i) \leftrightarrow \Gamma_1(o)$, if and only if, $FS_1 \models \Gamma_2(i) \leftrightarrow \Gamma_2(o)$, and because $F_{\Delta 1} \supseteq F_{\Delta 2}$, we have that $FS_1 \models \Gamma_2(i) \leftrightarrow \Gamma_2(o) \iff FS_1 \cap F_{\Delta 2} \models \Gamma_2(i) \leftrightarrow \Gamma_2(o)$. Therefore, equation (6.8) is equivalent to (6.9), and $fm_{\Delta 1} \sqsubseteq fm_{\Delta 2}$.

Now, let us consider behavioural refinement. Assume that for all $FS \in [\![fm_1]\!]^{F_1}$,

$$\mathcal{R}^{12}_{FS} \subseteq St_1 \times St_2, \; with \; (s_{0_1}, s_{0_2}) \in \mathcal{R}^{12}_{FS}$$

is the refinement relation for $A_1 \preceq A_2$.

We want to show that for all $FS \in [\![fm_{\Delta 1}]\!]^{F_{\Delta 1}}$, there exists a relation $\mathcal{R}^{\Delta 1 \Delta 2}_{FS}$, such that $(s_{0_{\Delta 1}}, s_{0_{\Delta 2}}) \in \mathcal{R}^{\Delta 1 \Delta 2}_{FS}$, and each $(s_{\Delta 1}, s_{\Delta 2}) \in \mathcal{R}^{\Delta 1 \Delta 2}_{FS}$ satisfies the conditions in Definition 6.2.3 as follows.

For each $FS \in [\![fm_{\Delta 1}]\!]^{F_{\Delta 1}}$, we have that:

(delay) if there exists some $\langle l_{\Delta 1}, \eta^{\Delta 1} \rangle \stackrel{\tau^*}{\Longrightarrow}{}^{d}_{[\![\Delta \mathcal{A}_1]\!] \downarrow_{FS}} \langle l'_{\Delta 1}, \eta^{\Delta 1'} \rangle$, we have to consider two cases regarding the nature of this transition in $\mathcal{A}_1 \downarrow_{FS}$[2]. That is, this transition can correspond to two kinds of (sequences of) transitions: 1) *includes, if any, only internal transitions* – thus, the same transition existed in $[\![\mathcal{A}_1]\!] \downarrow_{FS}$; and 2) *includes at least one transition with i and o in its set of actions* – thus, we can decompose this transition in smaller cases. In case 1), this transition corresponds to a (possible sequence of internal) transition(s) in $\mathcal{A}_1 \downarrow_{FS}$, and since these transitions are not affected by Δ, we have that there exists some

[2]Projection of IFTA is defined analogous to projection of IFTS

$\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle \stackrel{\tau^*}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_1]\!]\downarrow_{FS}} \langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle$. Then by $\mathcal{A}_1 \preceq \mathcal{A}_2$, we have that there exists some $\langle l_2, \eta^2\rangle \stackrel{\tau^*}{\Longrightarrow}{}^{d}_{[\![\mathcal{A}_2]\!]\downarrow_{FS}} \langle l'_2, \eta^{2'}\rangle$, such that $(\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle, \langle l_2, \eta^2\rangle) \in \mathcal{R}^{12}_{FS}$ and $(\langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle, \langle l'_2, \eta^{2'}\rangle) \in \mathcal{R}^{12}_{FS}$. Therefore, since the transition in $\mathcal{A}_2 \downarrow_{FS}$ consists only of internal transitions, if any, then there exists some $\langle l_2, \eta^2\rangle \stackrel{\tau^*}{\Longrightarrow}{}^{d}_{[\![\Delta\mathcal{A}_2]\!]\downarrow_{FS}} \langle l'_2, \eta^{2'}\rangle$, such that $(\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle, \langle l_2, \eta^2\rangle) \in \mathcal{R}^{\Delta 1\Delta 2}_{FS}$, and $(\langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle, \langle l'_2, \eta^{2'}\rangle) \in \mathcal{R}^{\Delta 1\Delta 2}_{FS}$. In case 2), this transition corresponds to a sequence of transitions in $\mathcal{A}_1 \downarrow_{FS}$, which contains internal transitions, if any. Let us assume that, without loss of generality, there is *only one* transition with i and o in its set of actions. That is, there exists some

$$l_{\Delta 1} \stackrel{\tau}{\Rightarrow}_{\mathcal{A}_1\downarrow_{FS}} l_i \xrightarrow{\{i,o\}}_{\mathcal{A}_1\downarrow_{FS}} l_j \stackrel{\tau}{\Rightarrow}_{\mathcal{A}_1\downarrow_{FS}} l'_{\Delta 1}$$

where $l \stackrel{\tau}{\Rightarrow}_{\mathcal{A}} l'$ is defined as a sequence of one or more internal transitions $l \xrightarrow{\tau}_{\mathcal{A}} l^1 \ldots l^n \xrightarrow{\tau}_{\mathcal{A}} l'$. Then, by $[\![\mathcal{A}_1]\!]$ there exists some

$$\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle \stackrel{\{i,o\}}{\Longrightarrow}{}^{d'}_{[\![\mathcal{A}_1]\!]\downarrow_{FS}} \langle l_j, \eta^{\Delta 1}_j\rangle \stackrel{d''}{\Longrightarrow}_{[\![\mathcal{A}_1]\!]\downarrow_{FS}} \langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle$$

such that $d = d' + d''$. Then, because $\mathcal{A}_1 \preceq \mathcal{A}_2$, and because $i, o \subseteq I_m \times O_m$ for $m = 1, 2$, this is, $\stackrel{\{i,o\}}{\Longrightarrow}{}^{d'}_{[\![\mathcal{A}_1]\!]\downarrow_{FS}}$ corresponds to an output transition, we have that there exists some

$$\langle l_2, \eta^2\rangle \stackrel{\{i,o\}}{\Longrightarrow}{}^{d'}_{[\![\mathcal{A}_2]\!]\downarrow_{FS}} \langle l_k, \eta^2_k\rangle \stackrel{d''}{\Longrightarrow}_{[\![\mathcal{A}_2]\!]\downarrow_{FS}} \langle l'_2, \eta^{2'}\rangle$$

such that $(\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle, \langle l_2, \eta^2\rangle) \in \mathcal{R}^{12}_{FS}$, $(\langle l_j, \eta^{\Delta 1}_j\rangle, \langle l_k, \eta^2_k\rangle) \in \mathcal{R}^{12}_{FS}$ and $(\langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle, \langle l'_2, \eta^{2'}\rangle) \in \mathcal{R}^{12}_{FS}$. Then, by definition of Δ, there exists some

$$\langle l_2, \eta^2\rangle \stackrel{\{i,o\}}{\Longrightarrow}{}^{d}_{[\![\Delta\mathcal{A}_2]\!]\downarrow_{FS}} \langle l'_2, \eta^{2'}\rangle$$

such that $(\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle, \langle l_2, \eta^2\rangle) \in \mathcal{R}^{\Delta 1\Delta 2}_{FS}$, and $(\langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle, \langle l'_2, \eta^{2'}\rangle) \in \mathcal{R}^{\Delta 1\Delta 2}_{FS}$.

(output) if there exists some $\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle \stackrel{OI}{\Longrightarrow}{}^{d}_{[\![\Delta\mathcal{A}_1]\!]\downarrow_{FS}} \langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle$, we can decompose this transition as

$$\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle \stackrel{\tau^*}{\Longrightarrow}{}^{d}_{[\![\Delta\mathcal{A}_1]\!]\downarrow_{FS}} \langle l_j, \eta^{\Delta 1}_j\rangle \xrightarrow{IO}_{[\![\Delta\mathcal{A}_1]\!]\downarrow_{FS}} \langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle$$

Since we already proved the *delay* case, we know that there exists some $\langle l_{\Delta 2}, \eta^{\Delta 2}\rangle \stackrel{\tau^*}{\Longrightarrow}{}^{d}_{[\![\Delta\mathcal{A}_2]\!]\downarrow_{FS}} \langle l_k, \eta_k^{\Delta 2}\rangle$ such that $(\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle, \langle l_{\Delta 2}, \eta^{\Delta 2}\rangle) \in \mathcal{R}^{12}_{FS}$, and $(\langle l_j, \eta_j^{\Delta 1}\rangle, \langle l_k, \eta_k^{\Delta 2}\rangle) \in \mathcal{R}^{12}_{FS}$. Thus, we just need to consider the observable transition and two possible cases regarding this transition in automata $\mathcal{A}_1 \downarrow_{FS}$: 1) *corresponds to a transition in* $\mathcal{A}_1 \downarrow_{FS}$ – i.e., the same transition exists in $\mathcal{A}_1 \downarrow_{FS}$; and 2) *corresponds to an updated transition in* $\mathcal{A}_1 \downarrow_{FS}$ – i.e. it corresponds to a transition label with the actions $IO \cup \{i, o\}$. In case 1), there exists some $\langle l_j, \eta_j^{\Delta 1}\rangle \xrightarrow[[\![\mathcal{A}_1]\!]\downarrow_{FS}]{OI} \langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle$ and, by $\mathcal{A}_1 \preceq \mathcal{A}_2$, we know that there exists some $\langle l_k, \eta_k^{\Delta 2}\rangle \xrightarrow[[\![\mathcal{A}_2]\!]\downarrow_{FS}]{OI} \langle l'_{\Delta 2}, \eta^{\Delta 2'}\rangle$ such that $(\langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle, \langle l'_{\Delta 2}, \eta^{\Delta 2'}\rangle) \in \mathcal{R}^{12}_{FS}$. Therefore, there exists some $\langle l_k, \eta_k^{\Delta 2}\rangle \xrightarrow[[\![\Delta 2]\!]\downarrow_{FS}]{OI} \langle l'_{\Delta 2}, \eta^{\Delta 2'}\rangle$, such that $(\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle, \langle l_{\Delta 2}, \eta^{\Delta 2}\rangle) \in \mathcal{R}^{\Delta 1\Delta 2}_{FS}$, $(\langle l_j, \eta_j^{\Delta 1}\rangle, \langle l_k, \eta_k^{\Delta 2}\rangle) \in \mathcal{R}^{\Delta 1\Delta 2}_{FS}$, and $(\langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle, \langle l'_{\Delta 2}, \eta^{\Delta 2'}\rangle) \in \mathcal{R}^{\Delta 1\Delta 2}_{FS}$. The reasoning is analogous for case 2).

(input) if there exists some $\langle l_{\Delta 2}, \eta^{\Delta 2}\rangle \stackrel{IO}{\Longrightarrow}{}^{d}_{[\![\Delta\mathcal{A}_2]\!]\downarrow_{FS}} \langle l'_{\Delta 2}, \eta^{\Delta 2'}\rangle$, we need to consider two cases in $\mathcal{A}_2 \downarrow_{FS}$ 1) *corresponds to a transition in* $\mathcal{A}_2 \downarrow_{FS}$ – i.e. the same (sequence of) transition(s) exists in $\mathcal{A}_2 \downarrow_{FS}$; and 2) *corresponds to an updated transition in* $\mathcal{A}_2 \downarrow_{FS}$ – i.e. it corresponds to a (sequence of) transition(s) where some of the transitions had the actions i and o in its set of actions. Since case 1) is trivial, we focus in case 2). Let us assume, without loss of generality, it corresponds to a sequence of transitions in $\mathcal{A}_2 \downarrow_{FS}$ where there is only one transition labelled by $\{i, o\}$, and that the observable transition is also labelled by $\{i, o\}$. This corresponds to the following transition in the semantic representation:

$$\langle l_{\Delta 2}, \eta^{\Delta 2}\rangle \stackrel{d'}{\Longrightarrow}_{[\![\mathcal{A}_2]\!]\downarrow_{FS}} \langle l_i, \eta_i^{\Delta 2}\rangle \xrightarrow[[\![\mathcal{A}_2]\!]\downarrow_{FS}]{\{i,o\}} \langle l'_i, \eta_{i'}^{\Delta 2}\rangle \stackrel{IOio}{\Longrightarrow}{}^{d''}_{[\![\mathcal{A}_2]\!]\downarrow_{FS}} \langle l'_{\Delta 2}, \eta^{\Delta 2'}\rangle$$

where, $d = d' + d''$. Then, by $\mathcal{A}_1 \preceq \mathcal{A}_2$, we have that there exists some

$$\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle \stackrel{d'}{\Longrightarrow}_{[\![\mathcal{A}_1]\!]\downarrow_{FS}} \langle l_j, \eta_j^{\Delta 1}\rangle \xrightarrow[[\![\mathcal{A}_1]\!]\downarrow_{FS}]{\{i,o\}} \langle l'_j, \eta_{j'}^{\Delta 2}\rangle \stackrel{IOio}{\Longrightarrow}{}^{d''}_{[\![\mathcal{A}_1]\!]\downarrow_{FS}} \langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle$$

such that $(\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle, \langle l_{\Delta 2}, \eta^{\Delta 2}\rangle) \in \mathcal{R}^{12}_{FS}$, $(\langle l_i, \eta_i^{\Delta 1}\rangle, \langle l_j, \eta_j^{\Delta 2}\rangle) \in \mathcal{R}^{12}_{FS}$, $(\langle l'_i, \eta_{i'}^{\Delta 1}\rangle, \langle l'_j, \eta_{j'}^{\Delta 2}\rangle) \in \mathcal{R}^{12}_{FS}$, and $(\langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle, \langle l'_{\Delta 2}, \eta^{\Delta 2'}\rangle) \in \mathcal{R}^{12}_{FS}$. Then by definition of synchronization, we conclude that there exists some

$$\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle \stackrel{IO}{\Longrightarrow}{}^{d}_{[\![\Delta\mathcal{A}_1]\!]\downarrow_{FS}} \langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle$$

such that $(\langle l_{\Delta 1}, \eta^{\Delta 1}\rangle, \langle l_{\Delta 2}, \eta^{\Delta 2}\rangle) \in \mathcal{R}^{\Delta 1\Delta 2}_{FS}$, and $(\langle l'_{\Delta 1}, \eta^{\Delta 1'}\rangle, \langle l'_{\Delta 2}, \eta^{\Delta 2'}\rangle) \in \mathcal{R}^{\Delta 1\Delta 2}_{FS}$.

Therefore, $\Delta_{i,o}\mathcal{A}_1 \preceq \Delta_{i,o}\mathcal{A}_2$. □

Let us consider again the IFTA PP composed with the router R from Figure 5.4. If we want to check if we can replace PP by PP' (Figure 6.3) in the system composed by PP and R, because refinement is compositional, instead of checking if

$$PP' \bowtie_{(o_1,paypal)} R \preceq PP \bowtie_{(o_1,paypal)} R$$

it suffices to verify the following conditions:

1. $F_{PP} \cap F_R = F_{PP'} \cap F_R = \emptyset$
2. $fm_{PP'} \rightarrow ((\Gamma_{CM'}(o_1) \leftrightarrow \Gamma_{CM'}(paypal)) \leftrightarrow (\Gamma_{CM}(o_1) \leftrightarrow \Gamma_{CM}(paypal)))$
3. $PP' \preceq PP$

Conditions 1 and 2 correspond to the precondition for refinement to be a congruence with $\times$ and Δ, respectively. In our example, all conditions are satisfied. However, let us assume now that we have an IFTA PP'' which differs from PP only by changing the feature expression associated to the transition labelled with *paypal* from *pp* to $pp \vee op$ where *op* represents the support for online payment. In this case, when we try to replace PP by PP'' condition 2 does not hold. This is because in PP, *paypal* appears only when *pp* is present, however in PP'' *paypal* can appear when *op* is present and *pp* is absent. Thus, the resulting composed system with PP'' and R, models a concrete automaton that enables a synchronization between o_1 and *paypal* that was not possible before.

6.3 Variability-aware Refinement

The same way in which we defined a bisimulation relation between IFTS (Section 5.2.4) that takes into consideration the variability encoded in a single model, we can define a refinement relation over IFTS that it is also variability aware. In order to do this, we first define the feature expression of a sequence of transitions.

6.3.1 Definition (Feature expression of $\Rightarrow$). Given an IFTS $S = (St, s_0, A, T, fm, F, \gamma, \Gamma)$, we define the feature expression of a sequence of transitions, as follows

$$\widehat{\gamma}(s \overset{\omega}{\Rightarrow}^{d} s') = \gamma(s \xrightarrow{\omega_0} s_0) \wedge \gamma(s_0 \xrightarrow{\omega_1} s_1) \wedge \ldots \wedge \gamma(s_{n-1} \xrightarrow{\omega_n} s_n) \wedge \gamma(s_n \xrightarrow{\omega} s')$$

if $s \overset{\omega}{\Rightarrow}^{d} s' = s \xrightarrow{\omega_0} s_0 \xrightarrow{\omega_1} s_1 \ldots s_{n-1} \xrightarrow{\omega_n} s_n \xrightarrow{\omega} s'$, and such that for $i = 0 \ldots n$, $\omega_i \in \tau \uplus \mathbb{R}_{\geq 0}$, and if $\omega \in \mathbb{R}_{\geq 0}$, then $(\sum_{\omega_i \in \mathbb{R}_{\geq 0}} \omega_i) + \omega = d$, and if $\omega \in 2^A$, then $(\sum_{\omega_i \in \mathbb{R}_{\geq 0}} \omega_i) = d$. ■

Now we can formalize the variability-aware refinement function that takes a pair of states and calculates the set of products for which the two states are in a refinement relation.

6.3.2 Definition (Variability-aware refinement). Given two IFTS S and T, such that $fm_S \sqsubseteq fm_T$, a variability-aware refinement for S and T is a function $\mathcal{R} : St_S \times St_T \to FE(F)$ such that

$$\mathcal{R}(s,t) = \Big(\bigwedge_{s \overset{\tau^*}{\Longrightarrow}{}^{d}_{S} s'} \Big(\widehat{\gamma}(s \overset{\tau^*}{\Longrightarrow}{}^{d}_{S} s') \Rightarrow \bigvee_{t \overset{\tau^*}{\Longrightarrow}{}^{d}_{T} t'} \big(\widehat{\gamma}(t \overset{\tau^*}{\Longrightarrow}{}^{d}_{T} t') \wedge \mathcal{R}(s',t') \big) \Big) \Big) \wedge$$

$$\wedge \Big(\bigwedge_{s \overset{OIs}{\Longrightarrow}{}^{d}_{S} s'} \Big(\widehat{\gamma}(s \overset{OIs}{\Longrightarrow}{}^{d}_{S} s') \Rightarrow \bigvee_{t \overset{OIs}{\Longrightarrow}{}^{d}_{T} t'} \big(\widehat{\gamma}(t \overset{OIs}{\Longrightarrow}{}^{d}_{T} t') \wedge \mathcal{R}(s',t') \big) \Big) \Big) \wedge$$

$$\wedge \Big(\bigwedge_{t \overset{IO}{\Longrightarrow}{}^{d}_{T} t'} \Big(\widehat{\gamma}(t \overset{IO}{\Longrightarrow}{}^{d}_{T} t') \Rightarrow \bigvee_{s \overset{IO}{\Longrightarrow}{}^{d}_{S} s'} \big(\widehat{\gamma}(s \overset{IO}{\Longrightarrow}{}^{d}_{S} s') \wedge \mathcal{R}(s',t') \big) \Big) \Big)$$

where Is is either $Is = \emptyset$ or $Is \subseteq I_T$. Then, given a valid product $Fs \in [\![fm_S]\!]^{Fs}$, S refines T for product Fs, if and only if, $Fs \models \mathcal{R}(s_0,t_0)$. Let $\mathcal{R}_{max}$ be the largest refinement for S and T, then $\mathcal{R}_{max}(s_0,t_0)$ encodes the set of products for which S and T are bisimilar, where by largest it refers to the fact that given any other refinement $\mathcal{R}(s_0,t_0)$, we have that $\mathcal{R}(s_0,t_0) \Rightarrow \mathcal{R}_{max}(s_0,t_0)$. It should be notice that this set of products must be considered in the context of the feature model, this is, it is possible that some feature configuration for which S refines T is not allowed by the feature model. ■

6.3.1 Theorem. *Given two IFTS, S and T, such that $fm_S \sqsubseteq fm_T$, and a valid product $Fs \in [\![fm_S]\!]^{Fs}$, $Fs \models Rmax(s_0,t_0) \Leftrightarrow S \downarrow_{Fs} \preceq T \downarrow_{Fs}$.*

Proof. Let us first consider ($\Rightarrow$). If $Fs \models \mathcal{R}(s_0,t_0)$ we have that

$$\text{for all } s_0 \overset{\tau^*}{\Longrightarrow}{}^{d}_{S} s' \text{ there exists some } t_0 \overset{\tau^*}{\Longrightarrow}{}^{d}_{T} t'$$

such that

$$Fs \models \mathcal{R}(s',t') \text{ and if } Fs \models \widehat{\gamma}(s_0 \overset{\tau^*}{\Longrightarrow}{}^{d}_{S} s') \text{ then } Fs \models \widehat{\gamma}(t_0 \overset{\tau^*}{\Longrightarrow}{}^{d}_{S} t')$$

In other words, for all transitions $s_0 \overset{\tau^*}{\Longrightarrow}^d_S s'$ that exist in a product Fs, i.e., exist in $S \downarrow_{Fs}$, there is at least a transition $t_0 \overset{\tau^*}{\Longrightarrow}^d_T t'$ that exist in $T \downarrow_{Fs}$, and because $Fs \models \mathcal{R}(s', t')$, then this is also valid for transitions with s' and t' as origin states. This is exactly condition 6.1 in Definition 6.2.3. The reasoning is analogous for the other two cases, i.e., transitions labelled with OIs and IO. Thus, we conclude that $(s_0, t_0) \in \mathcal{R}_{Fs}$, where $\mathcal{R}_{Fs}$ is a refinement relation for $S \downarrow_{Fs} \preceq T \downarrow_{Fs}$.

Let us now consider ($\Leftarrow$). If $S \downarrow_{Fs} \preceq T \downarrow_{Fs}$ for a valid product $Fs \in [\![fm_S]\!]^{Fs}$, then let $\mathcal{R}_{Fs}$ be a refinement relation for $S \downarrow_{Fs} \preceq T \downarrow_{Fs}$, such that $(s_0, t_0) \in \mathcal{R}_{Fs}$. As before, this means that for all $(s, t) \in \mathcal{R}_{Fs}$, including (s_0, t_0), we have that

$$\text{for all } s \overset{\tau^*}{\Longrightarrow}^d_{S\downarrow_{Fs}} s' \text{ there exists some } t \overset{\tau^*}{\Longrightarrow}^d_{T\downarrow_{Fs}} t' \text{ such that } (s', t') \in \mathcal{R}_{Fs}$$

and since this is a projection over Fs, we have that

$$\text{if } Fs \models \widehat{\gamma}(s_0 \overset{\tau^*}{\Longrightarrow}^d_S s') \text{ then } Fs \models \widehat{\gamma}(t_0 \overset{\tau^*}{\Longrightarrow}^d_S t')$$

The reasoning for the other two cases from Definition 6.2.3 is analogous. Thus, we have that $Fs \models \mathcal{R}(s_0, t_0)$. □

6.4 Discussion

First we proposed a refinement relation for families of timed automata modelled as Interface Featured Timed Automata. Since each IFTA can be seen as: 1) a feature model, which determines a set of valid feature combinations; and 2) a set of concrete automata, where each of the concrete automata is determined by a valid set of features; we separated the notion of IFTA refinement into *variability refinement* and *behavioral refinement*. Furthermore, we decomposed IFTA resorting to other formalisms and define the corresponding notion of refinement, namely Interface Featured Transition System (IFTS) and Interface Transition Systems (ITS).

Additionally, we proposed a refinement function over IFTS that takes into account the variability encoded in the model. Thus it is possible to verify over a single model, the set of products for which an IFTS refines another.

The refinement relation proposed here is a pre-order and congruent with respect to IFTA product and synchronization, meaning refinement is compositional. However, in order to be congruent stronger conditions are required to hold. In particular, the implementation can only replace the specification in a composed environment, if 1) neither the implementation nor the specification may share features with the environment; and 2) the implementation does not add new interface connections and maintains all connections of the specification.

Although the requirement of not allowing new connections and maintain existing ones is reasonable, it can be too strict. For example, in alignment with the notion of ITS refinement, which allows to incorporate new behavior through new inputs, it might be desirable to incorporate new behavior in terms of new features. This way the example of PP'', introduced in Section 6.2.3, can be considered a refinement of PP. In fact, in [53], de Alfaro et. al. requires only that no new connections with the environment are made, while some connections can be lost. However, this is not sufficient to ensure that IFTA refinement is compositional. In this sense, as future work we would like to explore and formalize other notions of refinement and how these can affect the properties that one can expect from a refinement. For example, in the case of behavioural refinement, we could have defined that $\mathcal{A}$ refines $\mathcal{B}$, if and only if, for every feature selection Fs in $fm_{\mathcal{B}}$ (instead of $fm_{\mathcal{A}}$), $[\![\mathcal{A}]\!] \downarrow_{Fs} \preceq [\![\mathcal{B}]\!] \downarrow_{Fs}$. The advantage is that we can now incorporate behaviour in terms of new features. However, on the one hand, this requires that $fm_{\mathcal{A}}$ contains at least all feature selections allowed by $fm_{\mathcal{B}}$, meaning $fm_{\mathcal{A}}$ can not incorporate mandatory features. On the other, it can be too flexible, since we can not account for how the system will behave for new variability.

Chapter 7

Variability and Coordination

In Section 4.3 we presented the $\mathcal{R}$eo coordination language. A main advantage of $\mathcal{R}$eo is that synchronization is propagated through connectors by composition. Thus, we can use very simple primitive channels and nodes to build large complex coordination protocols. In alignment with our work, we would like as well to have *variability* being propagated through connectors by means of composition. The aim is to define variable and generic coordination protocols that adapt to the variability of the components they coordinate simply by linking ports that are variable. Towards this goal, we model $\mathcal{R}$eo connectors as IFTA.

Intuitively, given a set of components with variable ports, we want to define a coordination protocol for the entire family parametrized by features, such that different feature selections determine the different protocols. The protocols themselves merely define how variability propagates through their ends, while the actual variability is given by the variability of the components being coordinated. The composition mechanism for IFTA determines what are the valid feature selections, if any, for the protocol and for the entire composed system.

Since components have their own variability, the propagation could either be successful, implying that there is at least one product derivable from the composed system; or could fail due to a contradiction on the variability constraints of the components being coordinated. In the latter case, the envisaged set of components cannot be coordinated in such a way and thus there is no product derivable from the resulting composed system.

Chapter organization. Section 7.1 presents two approaches to model some commonly used $\mathcal{R}$eo connectors as IFTA. Section 7.2 illustrates with an example how variable connectors from both approaches are combined to model a complex coordination protocol. Finally, Section 7.3 discusses some benefits of this semantics and some

challenges faced when defining variable connectors to orchestrate families of components.

7.1 Variable $\mathcal{R}$eo Connectors

Various approaches can be designed to model $\mathcal{R}$eo connectors as IFTA. Each of them provides connectors with different degrees of variability. We describe two possibilities: a *conservative* approach in which a connector is present in a product only if all of its ports are present, and is not present if some port is absent; and a more *relaxed* one in which connectors allow some of their ports to be absent. The latter approach allows higher degree of variability, however, it can easily increase the number of products modelled by the SPL beyond what it is actually intended by the modeller, requiring the manual inclusion of variability restrictions upon the resulting composed feature model.

The type of approach used with each connector depends on the expected dependability between the components being coordinated. In practice often both approaches are used. We discuss this in more detail, illustrating with an example, in Section 7.2.

7.1.1 The Conservative Approach

In the conservative approach each connector behaves as it usually does in $\mathcal{R}$eo only if all of its ports are present in a given product. In case some port is absent, so is the entire connector. This is useful when the intention is to orchestrate components that should all depend on each other in every product.

When modelling generic connectors, their variability is defined in terms of a single unique feature. Figure 7.1 shows how the $\mathcal{R}$eo connectors introduced in Section 4.3 are modelled as IFTA using a conservative approach. For example, in the case of the `Merger`, we assign a feature $f_{i_1 i_2 o}$ to each transition. Its feature model, $fm = \top$, allows this feature to be either present or absent. When the connector is composed and the ports are synchronized with other components' ports, the composition of IFTA creates the necessary variability dependencies, i.e. the components become all present or all absent.

The `Sync` connector automata behaves as the *identity* when composed with other automata. This is formally captured by the following property.

7.1.1 Proposition (`Sync` behaves as identity). Given any IFTA $\mathcal{A}$ and the IFTA of the `Sync` connector, we have that $\Delta_{i,a}(\mathcal{A} \times \mathsf{Sync}(i, o)) \sim \mathcal{A}[o/a]$, if $\{i, o\} \not\subseteq A_{\mathcal{A}}$, and $a \in A_{\mathcal{A}}$, where $\mathcal{A}[o/a]$ is $\mathcal{A}$ with all occurrences of a replaced by o.

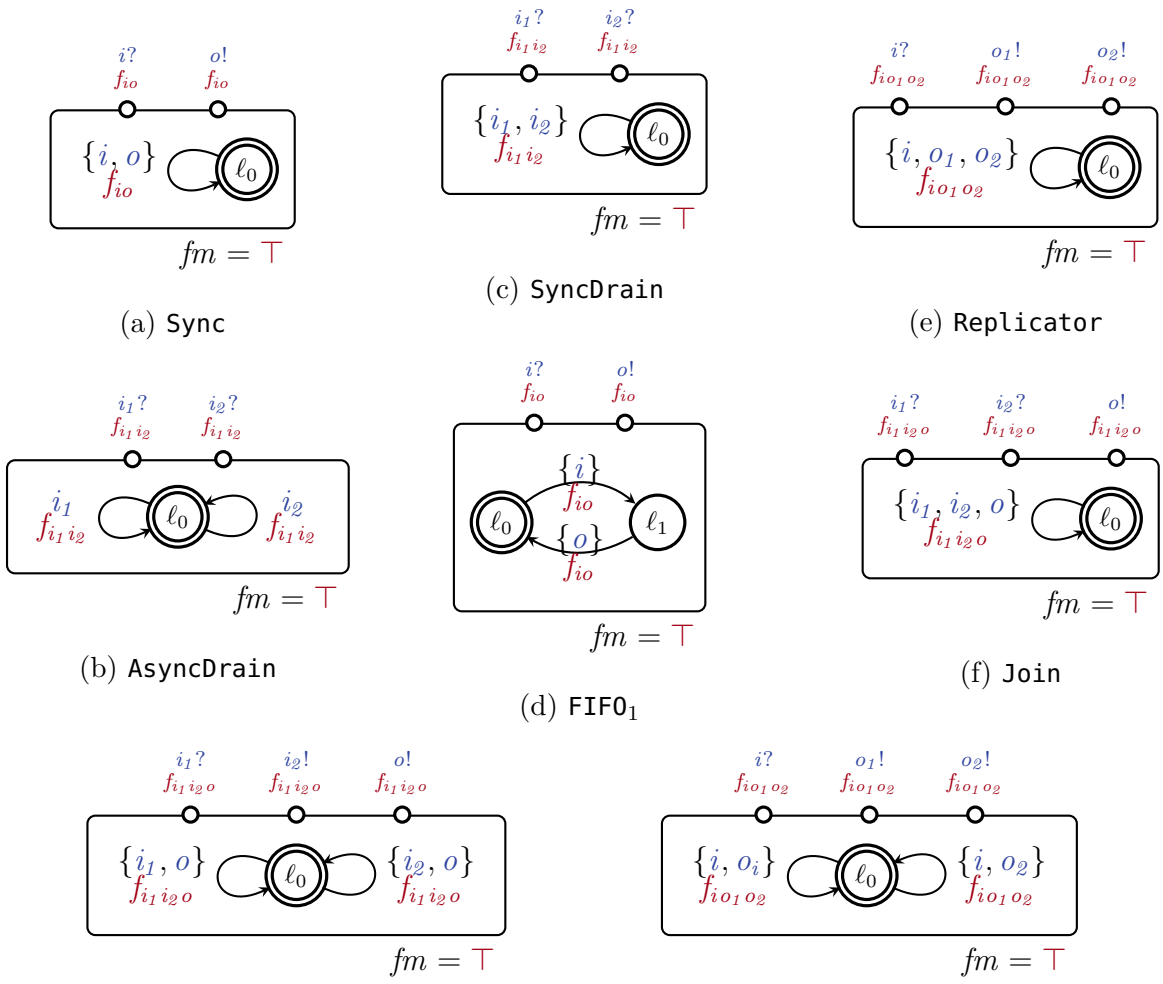


Figure 7.1: Example of $\mathcal{R}$eo connectors modelled as IFTA using the conservative approach.

Proof. First, let us notice that because of how we define bisimulation between IFTA, we need to make the following updates to prove bisimilarity:

- $fm_{\mathcal{A}[o/a]} = fm_{\mathcal{A}} \wedge (f_{io} \leftrightarrow \Gamma_{\mathcal{A}}(a))$
- $\gamma_{\mathcal{A}[o/a]}(\ell \xrightarrow{g,\omega,r}_{\mathcal{A}[o/a]} \ell') = \gamma_{\mathcal{A}}(\ell \xrightarrow{g,\omega[a/o],r}_{\mathcal{A}} \ell') \wedge f_{io}$, if $o \in \omega$
- $F_{\mathcal{A}[o/a]} = F_{\mathcal{A}} \cup \{f_{io}\}$

– $\Gamma_{\mathcal{A}[o/a]}(o) = \Gamma_{\mathsf{Sync}}(o)$

However, in practice, this is not necessary, since in essence these updates are done on top of a counterfeit feature that would be added if $\mathcal{A}$ is composed with Sync, i.e. f_{io}, and the restrictions added do not affect the set of products allowed by $\mathcal{A}$, they simply extend the set of features with f_{io}.

For simplicity, let $\mathcal{A}_S = (\mathcal{A} \times \mathsf{Sync}(i,o))$, and $\mathcal{A}' = \Delta_{i,a}(\mathcal{A}_S)$. Let us note that the set of edges in $\mathcal{A}'$ is defined as follows

$$T_{\mathcal{A}'} = \{(\ell_1,\ell_0) \xrightarrow{g,\omega,r}_{\mathcal{A}_S} (\ell_1',\ell_0) \mid i \notin \omega \text{ and } a \notin \omega\} \cup \quad (1)$$

$$\{(\ell_1,\ell_0) \xrightarrow{g,\omega\setminus\{i,a\},r} (\ell_1',\ell_0) \mid (\ell_1,\ell_0) \xrightarrow{g,\omega,r}_{\mathcal{A}_S} (\ell_1',\ell_0) \text{ and } i \in \omega \text{ and } a \in \omega\} \quad (2)$$

where ℓ_0 is the initial and only location of Sync. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be the underlying FTS of $\mathcal{A}'$ and $\mathcal{A}[o/a]$, and note that $\mathcal{R} = \{(\langle(\ell_1,\ell_0),\eta\rangle, \langle\ell_1,\eta\rangle) \mid \ell_1 \in S_{\mathcal{A}[o/a]}\}$ is a bisimulation relating $\mathcal{F}_1$ and $\mathcal{F}_2$. Let $(\langle(\ell_1,\ell_0),\eta\rangle, \langle\ell_1,\eta\rangle) \in \mathcal{R}$. The proof for delay transitions follows trivially from the fact that $Inv(\ell_1,\ell_0) = Inv(\ell_1)$ for all $\ell_1 \in S_{\mathcal{A}[o/a]}$.

Let us consider any action transition $\langle(\ell_1,\ell_0),\eta\rangle \xrightarrow{\omega} \langle(\ell_1',\ell_0),\eta'\rangle \in T_{\mathcal{F}_1}$. If it comes from an edge in (1), then there exists some $\ell_1 \xrightarrow{g,\omega,r} \ell_1' \in T_{\mathcal{A}}$ $s.t.$ $a \notin \omega$, thus there exists some $\langle\ell_1,\eta\rangle \xrightarrow{\omega} \langle\ell_1',\eta'\rangle \in T_{\mathcal{F}_2}$; if it comes from (2), then there exists some $\ell_1 \xrightarrow{g,\omega_1,r} \ell_1' \in T_{\mathcal{A}}$ $s.t.$ $a \in \omega_1$, thus there exists some $\langle\ell_1,\eta\rangle \xrightarrow{\omega_1[o/a]} \langle\ell_1',\eta'\rangle \in T_{\mathcal{F}_2}$, where $\omega = \omega_1 \cup \{i,o\} \setminus \{i,a\} = \omega[o/a]$. Conversely, if there exists some $\langle\ell_1,\eta\rangle \xrightarrow{\omega} \langle\ell_1',\eta'\rangle \in T_{\mathcal{F}_2}$ and $o \notin \omega$, then there exists some $(\ell_1,\ell_0) \xrightarrow{g,\omega,r} (\ell_1',\ell_0) \in T_{\mathcal{A}_S}$ $s.t.$ $i \notin \omega \wedge a \notin \omega$, thus there exists some $\langle(\ell_1,\ell_0),\eta\rangle \xrightarrow{\omega} \langle(\ell_1',\ell_0),\eta'\rangle \in T_{\mathcal{F}_1}$; if $o \in \omega$, then there exists some $(\ell_1,\ell_0) \xrightarrow{g,\omega_1\cup\{o\}\setminus\{a\},r} (\ell_1',\ell_0) \in T_{\mathcal{A}'}$, such that $\omega = \omega_1[o/a] = \omega_1 \cup \{o\} \setminus \{a\}$, thus there exists some $\langle(\ell_1,\ell_0),\eta \xrightarrow{\omega} \langle(\ell_1',\ell_0),\eta'\rangle\rangle \in T_{\mathcal{F}_1}$.

In both cases, we have $\gamma_{\mathcal{F}_1}(\langle(\ell_1,\ell_0),\eta\rangle \xrightarrow{\omega} \langle(\ell_1',\ell_0),\eta'\rangle) = \gamma_{\mathcal{F}_2}(\langle\ell_1,\eta\rangle \xrightarrow{\omega} \langle\ell_1',\eta'\rangle)$. Furthermore, $fm'_{\mathcal{A}} = fm_{\mathcal{A}[o/a]}$. □

7.1.2 The Relaxed Approach

We propose now a more flexible model for $\mathcal{R}$eo connectors in which some ports might be absent. This sort of flexibility is useful when the components being coordinated are independent and do not always need to be present in the same product. By independent

we refer to the fact that they are variability independent, thus the components do not need to be present when the others components connected to the protocol are present, nor absent if the others are absent. If such were the case, the conservative approach is suitable to model the coordination protocol.

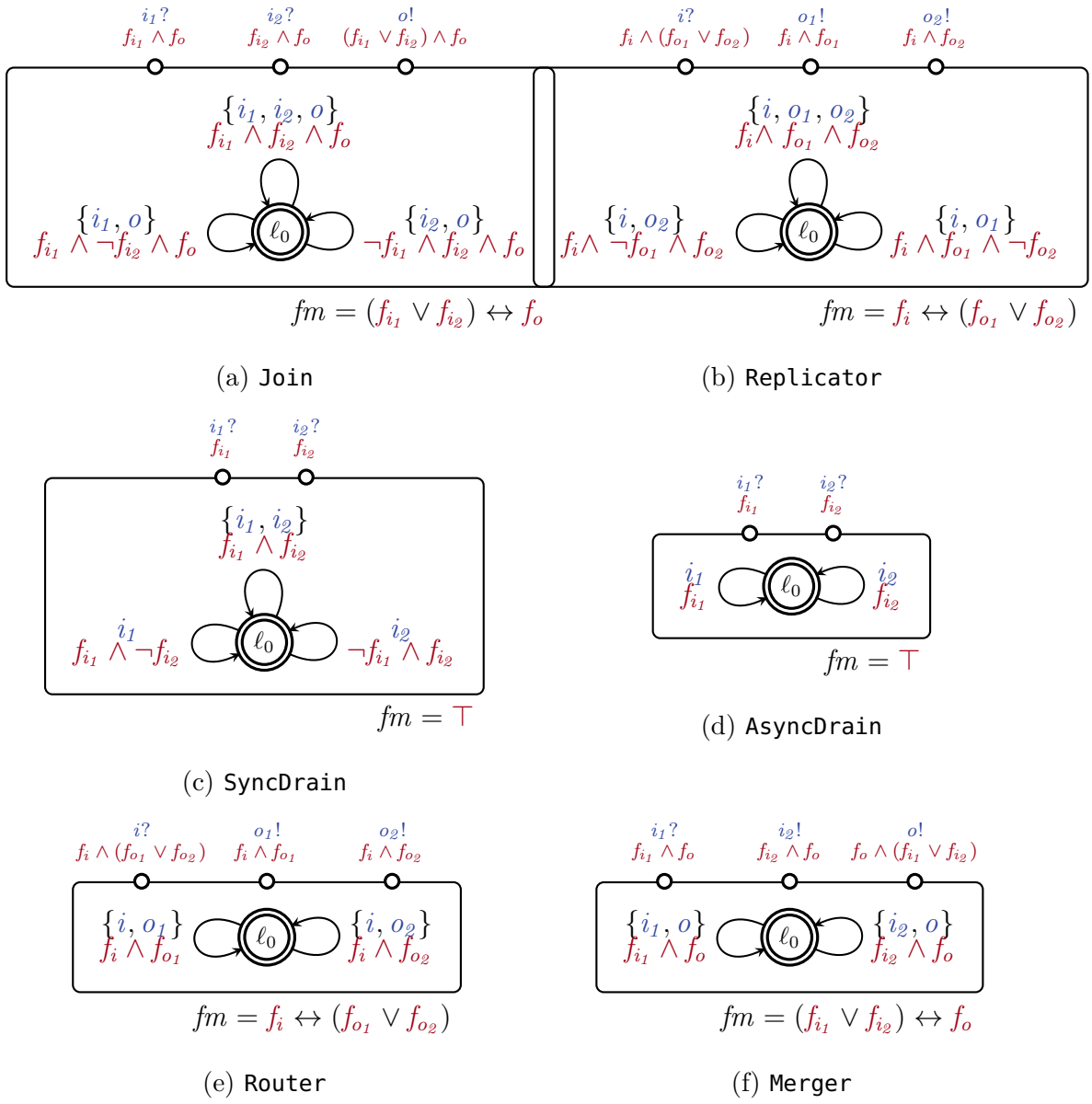


Figure 7.2: Example of $\mathcal{R}$eo connectors modelled as IFTA using the relaxed approach.

In practice, this only makes sense for some of the connectors introduced in Section 4.3. In particular, in the case of the Sync and $\texttt{FIFO}_1$ connector, it does no make sense that only one of the ports is present.

In the case of Sync, whose aim is to synchronize two components, such that information flows from the input to the output, if one end is missing, the connector should not be able to *create* outputs, nor receive and loose inputs. A similar reasoning applies to the $\texttt{FIFO}_1$ connector.

In this approach, the variability of each connector is defined in terms of a set of generic features: we assign a feature f_a for each port a of the connector and define its generic variability in terms of these features. Figure 7.2 shows the resulting models using a relaxed approach.

In all cases, connectors behave as their $\mathcal{R}$eo counterparts if all ports are present. In addition, they allow the following behaviour:

SyncDrain. It behaves as an asynchronous drain if one of its ends is missing. If both inputs are missing the entire connector is not present.

AsyncDrain. It always behaves as its $\mathcal{R}$eo counterpart in the sense that it accepts only one input at a time. The difference with the original and the conservative connector is that it does not requires both inputs to be present in a given product.

Join. It allows one of the inputs to be missing, in which case it behaves as a Sync, synchronizing the corresponding input with the output. If all inputs or the output are missing, so does the entire connector.

Merger. It always behaves as its $\mathcal{R}$eo counterpart in the sense it synchronizes only one of its inputs with the output. The difference lays in that it does not require both inputs to be present in a given product. In this sense, it encodes two Sync connectors. If all inputs or the output are missing, so does the entire connector.

Router. It always behaves as its $\mathcal{R}$eo counterpart in the sense it synchronizes the input with only one of its outputs. The difference again lays in that it does not require both outputs to be present in a given product. Thus, it encodes as well two possible Sync connectors. If all outputs or the input are missing, so does the entire connector.

Replicator. It allows one of the outputs to be absent, in which case it behaves as Sync, synchronizing the input with the corresponding output. If all outputs or the input are missing, so does the entire connector.

It is also possible to describe connectors using the relaxed approach and achieve the conservative approach just by tweaking the feature model. In practice, the most

flexible approach would be to define the variability of all ports to be independent from each other and allow the user to express restrictions through the feature model.

7.2 Example: Synchronous Merger

This section illustrates how to compose variable connectors to make a complex coordination protocol. We use the *synchronous merger* described in [148] and defined in [130] using $\mathcal{R}$eo connectors.

The original protocol coordinates two components *C1* and *C2*, each of which has an input port and an output port. The coordination is as follows: *C1* and *C2* can execute together, in which case the protocol waits for both components to produce an output, merging the outputs when ready; or only one component executes, in which case the protocol waits for the output of the corresponding component.

In our example, components *C1* and *C2* are variable, i.e. each one can be present or absent. Thus, the protocol needs to adapt to the presence or absence of such components. We assume that *C1* and *C2*, as well as their input and output ports, depend on a feature *c1* and *c2*, respectively. The feature model of the variable protocol should allow any combination of these features, including the absence of both.

The resulting model for this protocol is shown in Figure 7.3 and consists of 29 variable $\mathcal{R}$eo connectors: four `merger` (`>-`), three `router` (`Xor`), nine `replicator` (`-<`), seven $\texttt{FIFO}_1$ (`[]`), and six `syncdrain` (`>-<`). Out of these, there are 22 conservative connectors (represented by white boxes), two connectors using the relaxed approach (represented by light grey boxes), and five connectors using the relaxed approach with additional restrictions over their feature model (represented by dark grey boxes). There is one available input port, and one available output port that can be connected to the environment, i.e. other components, through which it is possible to invoke `C1` and `C2` and wait for their termination. They are represented by two ellipses labelled `in` and `out`, respectively. There are two additional boxes labelled `C1` and `C2`, representing the components being coordinated.

The resulting feature model allows four products, $fm = \{\{c1, c2\}, \{c1\}, \{c2\}, \{\}\}$. For simplicity, we only show features associated to the components *C1* and *C2*. However, the resulting feature model also contains the features corresponding to the connectors present. By selecting concrete products we can see how the protocol adapts to the presence or absence of the components. Figure 7.4 exemplifies this. In particular, Figure 7.4a shows the resulting model when only *C1* is present, while Figure 7.4b shows the resulting model when only *C2* is present. Black arrows indicate what connections between ports remain present in the product, while light grey arrows indicate lost connections.

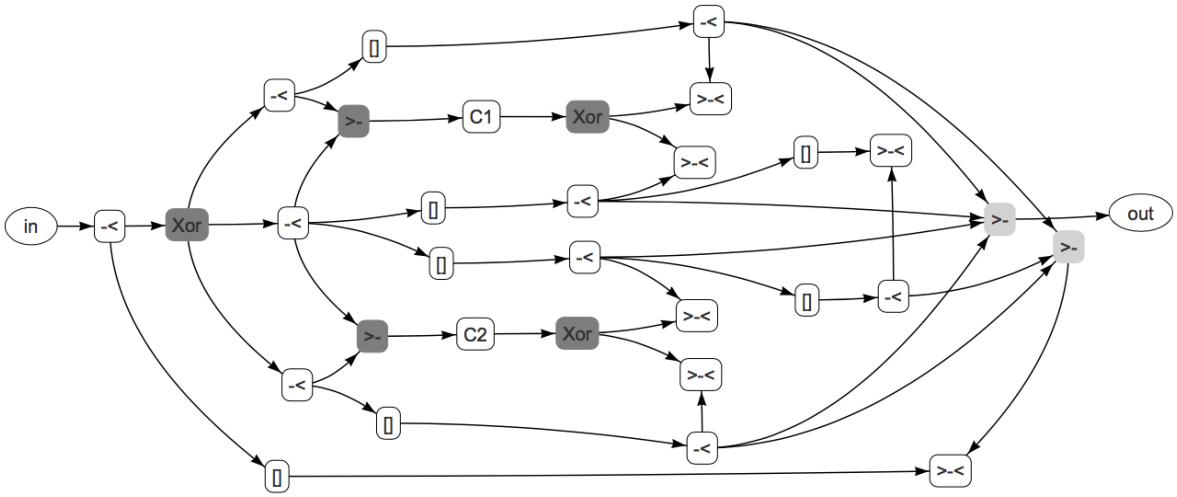
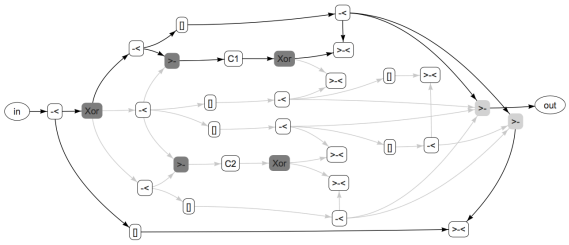


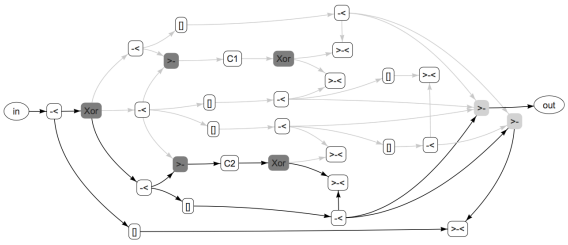
Figure 7.3: Synchronous merger with support for variable components

In order to accomplish a model with these four possible products it is necessary to use the right type of connector, and as it can be seen, even then it might be necessary to add additional restrictions to prevent undesired variability.

As mentioned, there are five relaxed connectors with additional restrictions over their feature models. For example, the `Router` (`Xor` center left) requires that its output port connected to the four-output replicator must be present if and only if *C1* and *C2* are present. Otherwise, because the router is relaxed, this output can be present or absent. In the latter case, the entire four-output replicator is absent, as well as all the conservative connectors it is connected to, propagating the absence of conservative connectors along the way until reaching the two relaxed mergers on the right side.



(a) Feature selection: $Fs = \{c1\}$



(b) Feature selection: $Fs = \{c2\}$

Figure 7.4: Two instantiations of the variable synchronous merger from Figure 7.3. Black arrows represent active port connections, while light grey arrows represent absent connections.

Figure 7.5 illustrates the projected model for a product with features *c1* and *c2* present. In this case components can execute in an exclusive manner, but never together.

Similar issues occur with the other connectors. The restriction imposed in these case are as follows:

- `Merger`(`>-` top left): it requires that its top input must be present if and only if its output is present.
- `Merger`(`>-` bottom left): it requires that its bottom input port must be present if and only if its output is present.
- `Router`(`Xor` top right): it requires that its top output must be present if and only if the input is present.
- `Router`(`Xor` bottom right): it requires that its bottom output must be present if and only if the input is present.

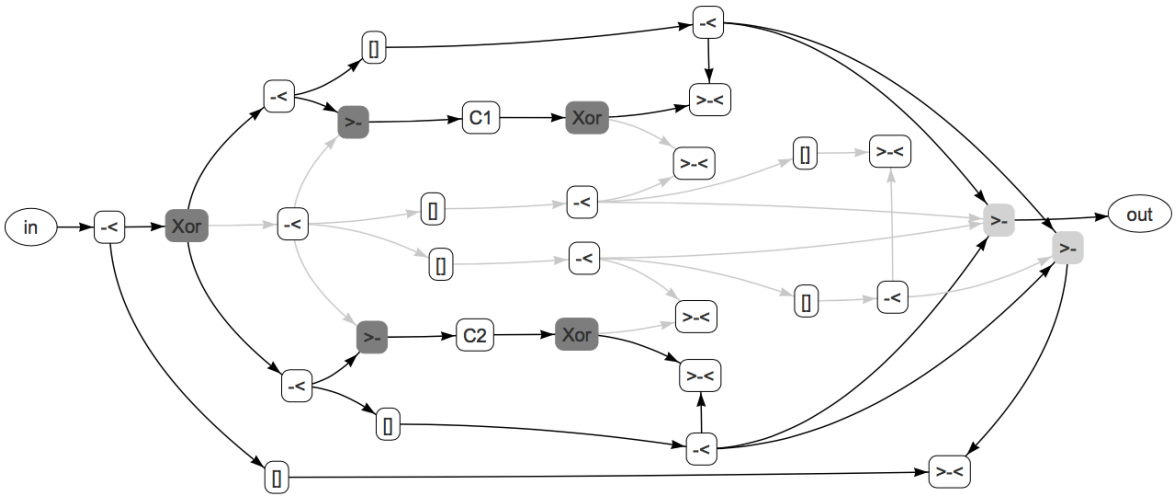


Figure 7.5: Example of an undesired projected product allowed when no additional restrictions are made over the variability model of the leftmost `Router`.

The figures shown here were automatically generated by a proof-of-concept prototype[1] developed as part of this thesis. In this case, the tool allows to visualize a network of IFTA, it calculates the valid products allowed by the composed system, and allows to select the different products, showing the corresponding projected architecture. The layout and the colours to distinguish the different types of approaches were adapted manually.

[1] https://github.com/haslab/ifta

7.3 Discussion

The initial intention was to defined variable connectors that quickly adapt to the variability of the components they coordinate. However, the example shown in Section 7.2 illustrates the complexity of modelling variable coordination mechanism and the need for more than one approach to deal even with basic connectors. Even then, it might be necessary to manually add restrictions to the feature model in order to reduce the variability allowed for the composed system.

It would not have been feasible to define the correct type of connectors, nor the variability restrictions, without having a tool to calculate all possible feature selections of the composed system, and to visualize all possible port connections in a given feature selection. As future work, we intend to work on other approaches to model $\mathcal{R}$eo connectors that may simplify even more the definition of these protocols in a way that takes away the burden from the user of defining additional restrictions. A possible path can be to work on a constraint solver to automatically suggest variability restrictions in order to achieve a given desire outcome.

Finally, it is possible to observe how defining multi-action transitions simplifies significantly the design of $\mathcal{R}$eo connectors in comparison with other approaches as shown in Section 5.1.

Chapter 8

A Virtual Factory Approach

This chapter introduces the idea of the proposed virtual factory approach. A virtual factory can be seen as a framework, populated with tools and guidelines, to automate and make more efficient the development of families of digital public services, particularly, in this thesis, we focus on a family of public transport licensing services, within the smart mobility domain. The virtual factory comprises artefacts contributing to three stages in the service development process, namely, planning, domain engineering and software engineering.

Chapter Organization. Section 8.1 describes the concept of a virtual factory. Then, Sections 8.2 to 8.4 discusses each component of the virtual factory, namely, the ones concern with planning, domain engineering, and software engineering, respectively. Finally, Section 8.5 discusses some advantages and limitations of the framework proposed.

8.1 Virtual factory

A virtual factory refers to a framework comprising software tools, procedures, guidelines, practices, models and other artefacts that assist different type of stakeholders – e.g., policy makers, government officials, and software engineers, to automate and make more efficient the development of a families of digital public services.

The idea of the virtual factory seeks to shift the paradigm from silo-based development of services to a component-based development, using software engineering techniques, particularly those related to SPLs, and by proactively identifying and taking advantage of common features, as well as commonalities on business processes present on different members of a family of services. The virtual factory seeks to contribute to the different stages in the development of a predetermined family of services, namely

planning, domain engineering and software engineering.

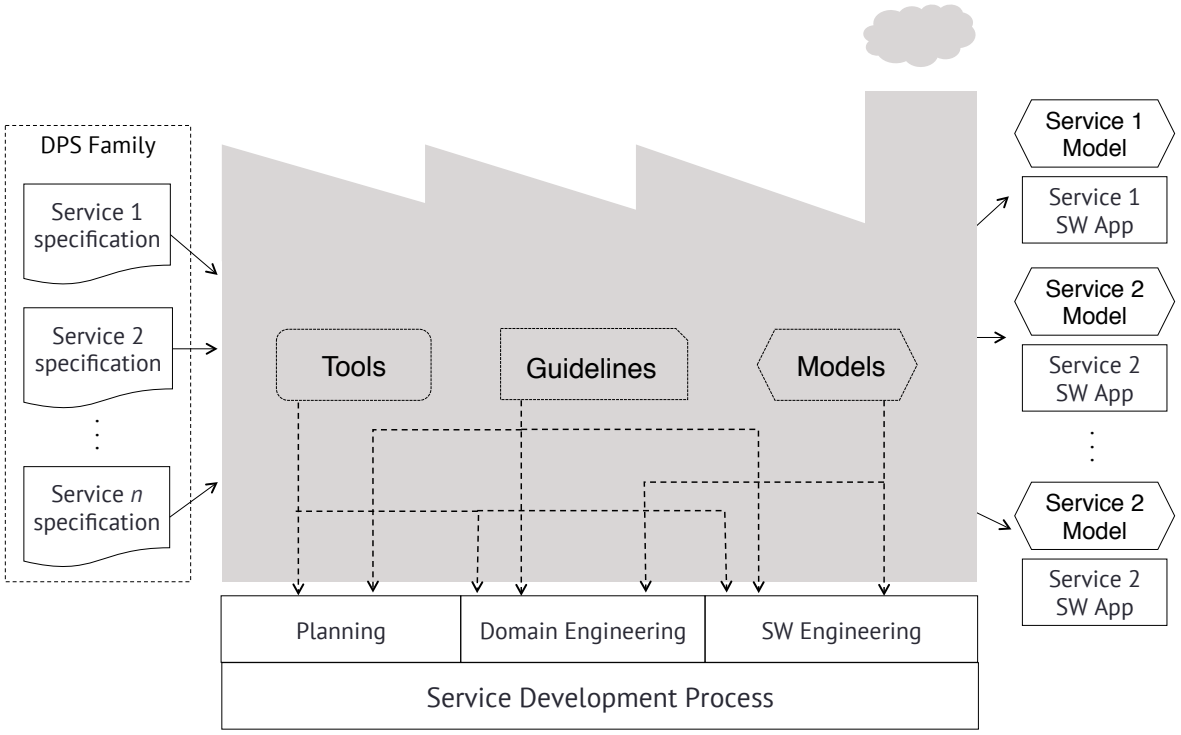


Figure 8.1: Virtual factory concept

Figure 8.1 shows a scheme of the virtual factory concept. The concept of the virtual factory relies on three principles: 1) service family specific – it is configurable for a given family, in the case of this thesis, for a family of licensing of public transport services; 2) scalable – it can be extended with other components; and 3) generalizable – it can be applicable to other families. Then, in the context of a specific family, stakeholders can specify an envisaged service, and automatically obtain, to some extent, concrete models and software applications for it. As in any SPL development process, this may require the development of additional features not supported by the family considered. The software applications should implement the requirements specified by the formal models. In addition, the models are used to verify if the derived application of a concrete service satisfies the specified requirements.

For the virtual factory proposed in this thesis, the current state of development includes the components shown in Figure 8.2. The figure depicts elements comprised for each of the stages. Grey boxes highlight contributions of the thesis, while white boxes represent existing concepts and tools on top of which we build such contributions.

The hexagon represents concrete models. The rest of the figures represent concrete information (e.g., specifications, properties, etc.) related to a family of services.

The virtual factory deals with different levels of abstraction. Firstly, it deals with the strategic planning of services in a given dimension of a smart city (planning). Then, it gradually focuses on a specific family, documenting the structural properties of the domain, for example by fixing the vocabulary (domain engineering). Finally, it focuses on specific elements of the structural domain and documents behavioural properties associated to such elements (software engineering).

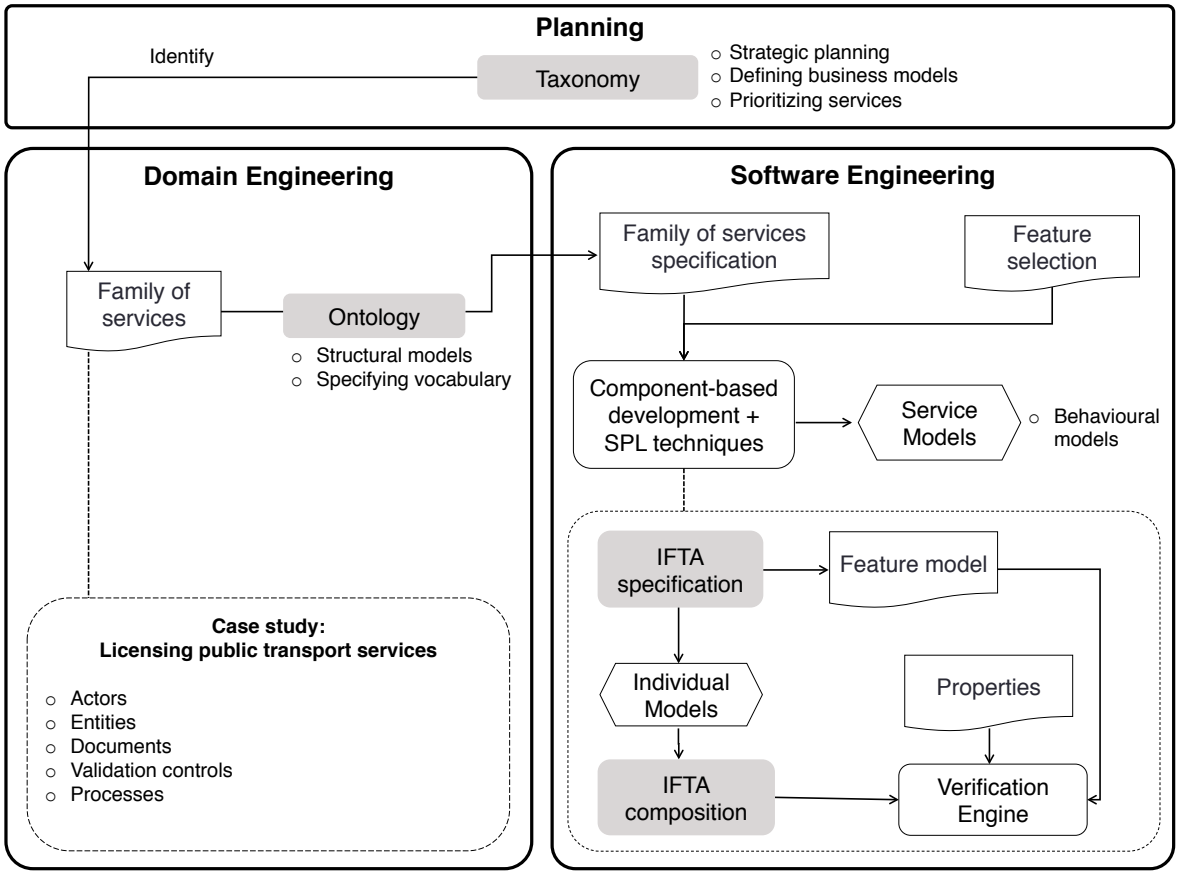


Figure 8.2: Components of the virtual factory

For the planning stage we provide the taxonomy of smart mobility services proposed in Chapter 3. The taxonomy serves as a tool for government practitioners involved in the planning and design of smart mobility services, by identifying stakeholders to whom to deliver services, types of services to be delivered, and public value delivered by such services, facilitating the definition of business models for developing initiatives. In

addition, the taxonomy can help in the identification of types of mobility services to be delivered at the city level. In fact, it contributed to the identification of a concrete family of services to serve as a case study to model families of services in a compositional way, in particular, a family of licensing public transport services.

For the domain engineering stage the virtual factory provides an ontology to capture the common structural vocabulary of the identified family. As discussed before, most of the services identified by the taxonomy of smart mobility services were developed by non-government entities or co-created with government. However, government must ensure the provision of public transport services as a basic service. The provision of licenses help government to ensure and regulate such provision. In addition, this family of services is provided by local government worldwide, and as such it is of interest and scalable to be reused by governments with different levels of resources and legal backgrounds. The ontology fixes a common vocabulary for the domain, by modelling structural elements, their attributes, and relationships.

Finally, for the software engineering stage, the virtual factory uses component-based development and SPL techniques to rapidly model families of services and to derive concrete models of members of the family. In particular, the virtual factory uses IFTA, the compositional formalism introduced in Chapter 5, to model behavioural aspects of the elements identified in the domain engineering stage. To illustrate the advantages of using a compositional formalism, IFTA, and the principles of exogenous coordination to rapidly model the behavioural aspects of a family of services, we present in this chapter models of behavioural aspects for the identified family. In addition, we briefly describe a proof-of-concept tool developed to specify, compose, visualize, and translate IFTA models to other formalisms.

The following sections present the components provided by the virtual factory for each development stage.

8.2 Planning

The taxonomy proposed in Chapter 3 provides a common vocabulary to describe, discuss, and share information about smart mobility initiatives. This is a first step towards a conceptual framework for smart mobility services supporting standardized information, and enabling knowledge sharing about smart mobility initiatives, which can address some of the challenges discussed in Section 3.6. The faceted structure of the taxonomy enables different stakeholders to recover information in ways that better suit their interests, e.g., recovering benefits or technology associated to a given type of service. In particular, we recognize three main potential users of the taxonomy: policy makers and government officials, IT staff, and researchers. Usage scenarios for each

type of user are discussed below and summarized in Figure 8.3.

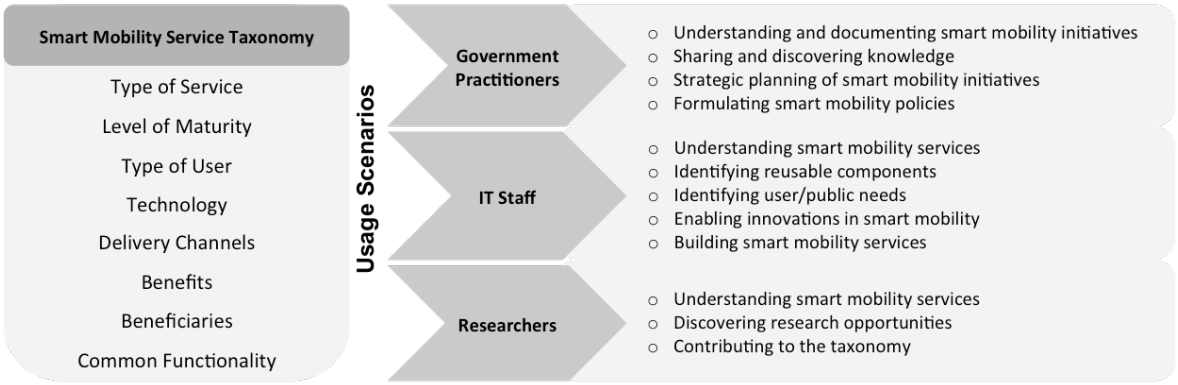


Figure 8.3: Usage scenarios of the taxonomy

Uses of the taxonomy by government practitioners – i.e. policy makers and government officials involved in the development of smart mobility initiatives include: 1) strategic planning and policy making – the taxonomy helps to identify stakeholders to whom services shall be delivered, to identify and illustrate different types of services to be delivered, and to identify corresponding benefits and beneficiaries, facilitating the justification of business cases for each initiative. For example, government practitioners motivated by a specific need can use the taxonomy to identify type of services that address those needs, benefits they provide, and functionality offered by the services. Conversely, given a type of service, government practitioners can use the taxonomy to understand the benefits that it delivers in order to convince different stakeholders involved to implement and use such a service; and 2) learning from others experiences – the taxonomy facilitates standardize documentation of initiatives, as well as the creation of a catalogue of such initiatives, whether successful or not.

Uses of the taxonomy for IT staff, whether in government or not, include: 1) identifying families of public services – the taxonomy specifies common functionality that can be used to develop reusable components for smart mobility services. Furthermore, it can be used to study and identify families of service, i.e. services with a high level of similarities in their functionality but differing in some aspects. Families of services can then be developed following techniques from Software Product Line Engineering (SPLE), simplifying development efforts, improving service interoperability, and reducing service development time; and 2) identifying innovation – the taxonomy can relate technologies used in different types of services. This is also useful to identify innovative uses of existing technology, which can serve to reduce costs.

Uses of the taxonomy by researchers include: 1) understanding the domain – the taxonomy provides a thorough view of different aspects of smart mobility services; and 2) development of new research lines – researches can use the taxonomy for discovering research opportunities, e.g., by identifying new dimensions or by evolving existing dimensions based on new technology trends and innovations.

Taking full advantage of the benefits discussed above requires an online knowledge base of smart mobility initiatives, with customizable searching tools, where each initiative is described using the dimensions and concepts proposed. In practice, it is challenging to achieve a unique global repository of initiatives. However, having a taxonomy that standardizes the domain, encourages various interested actors, mainly governments, in providing their own repositories. Thus, although information is not available in a unique place, it is consistent and structured, and facilitates (automatic) retrieval of information. In this sense, governments are considered the most interested stakeholders responsible for providing such online platforms.

8.3 Domain Engineering

We identified a family of licensing public transport services to pre-define and to serve as a case study for validation. As discussed in Chapter 1, this kind of services are present in most local governments and are required to regulate the provision of many public transport services.

The services in the family share common vocabulary and functionality and are amendable to be delivered through similar business processes. The domain engineering components of the virtual factory aim at fixing the general vocabulary, attributes, properties, and architectural schemes of such a domain.

In order to do this we include an ontology as one of the components for this stage. To develop the ontology, first, we selected two countries, Ireland and Portugal, to study services of licensing public bus passenger services, and we analysed how the services were delivered in both cases. We analysed government guidelines and application forms from both countries with the purpose of identifying: 1) licenses required for the provision of public bus transport services, 2) documentation required for the application of each license, 3) application process activities, and 4) entities involved in the provision of the licensing services.

Based on the analysis, we propose an ontology of public bus passenger services to capture the common vocabulary of the domain and to standardize knowledge. The proposed ontology captures common concepts – e.g. actors, supporting documents, and attributes required in the application and processing stage of three licenses: 1) a license to operate passenger services, 2) a license to provide a bus passenger service

across specified pick up and set down points following a predefined schedule and a fare scheme, and 3) a license for each vehicle used to transport passengers.

In the following subsections we describe the methodology used to build the ontology and the ontology itself.

8.3.1 Methodology

The research methodology comprises four activities explained below and it is illustrated in Figure 8.4.

- *Literature Review* – 1) assessing existing related work on the development of digital licensing services, and on the use of ontologies to support Digital Government, 2) identifying a family of licensing public transport services to serve as case study, sharing common vocabulary and functionality amendable to be delivered through similar business processes.

- *Domain Analysis* – to understand the licensing public transport service domain, in particular by studying government guidelines and application forms from two case studies of licensing public bus passenger services. The domain analysis produced UML Class and Activity Diagrams, contributing to identifying main domain elements and business processes used during the licensing application and processing stages.

- *Ontology Analysis* – studying methodologies and tools used to define ontologies and selecting a suitable approach to define an ontology for licensing public transport services. The background study on ontologies was described in [40].

- *Ontology Definition* – defining an ontology for licensing public transport services able to capture common vocabulary of the various services in the family analysed in the domain analysis, and using methodologies and tools selected from the ontology analysis activity.

8.3.2 Ontology

This section describes each of the steps applied to build the ontology of public transport licensing services.

We follow the Representation Formalism for Software Engineering Ontologies (REFSENO) methodology [142]. It is a representation formalism to model the structure of an experience base for software engineering. The motivation behind this formalism is to build ontologies to: collect experiences from software projects; capture and reuse

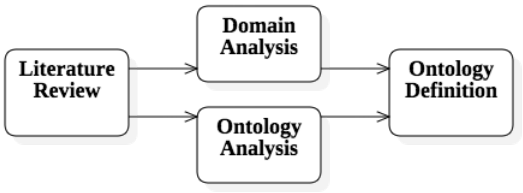


Figure 8.4: Methodology for ontology development.

explicit software development know-how; provide support for software organizations in collecting, packaging, validating and reusing experiences; and formalize informal knowledge. The methodology suggests a process model to develop ontologies using a set of pre-defined tables to structure knowledge, including tables for defining: a glossary of concepts, attributes of the concepts, relationships among concepts, and instances of the concepts to capture experience. The main advantage of REFSENO over other formalisms is 1) its support for similarity-based retrieval knowledge, and 2) a clear distinction between stable knowledge (concepts) and example knowledge (experience).

The following sections elaborate on each of the steps of the REFSENO process model and illustrate some of the tables developed during this process. The complete tables can be found in [39].

Ontology specification

The first step comprises specifying the ontology. This includes information about the domain being modelled, the purpose of the ontology, its scope, and relevant information regarding its authors, development date, and other data. Table 8.1 defines the ontology specification.

Glossary of concepts

The second step consists of defining all concepts identified in the scope of the ontology, as defined during the specification step. For this purpose the methodology proposes a table listing all concepts alphabetically with their definitions. Table 8.2 presents the glossary of concepts for some of the main concepts in the public transport licensing service ontology.

Concepts relationships

The third step consists of identifying semantic relationships between concepts. For this purpose, the methodology proposes a graphical notation using boxes for the concepts

Table 8.1: Ontology specification

Domain	Licensing Public Transport Services
Date	November, 2015
Conceptualized by	Guillermina Cledou, Elsa Estevez, Luis Barbosa
Purpose	To model required information when providing and requesting public transport licensing services in order to: 1) facilitate the transition from service delivery through traditional channels to electronic channels, 2) serve as a tool defining a common vocabulary to share knowledge and have a common understanding between domain experts and software engineers, and 3) be used as a supporting tool for the development of a SPL for the modelled domain.
Level of formality	Semi-formal
Scope	List of concepts: Additional Information, Appeal, Application Payment Receipt, Application Process Criteria, Approved License, Bus Stop Approval, Business Stakeholder, Criminal Record Certificate, Day Specific Schedule, Eligibility Criteria, Existing License, Financial Capability Evidence, Individual Stakeholder, Journey, Legal Person Card, License Application, License Application Supporting Documents, License Decision, License for Passenger Transport, License for Transport Operator, License for Vehicle, Life Cycle Stage per License, Livery, Map, Market Information, Ownership Certificate, Registration Certificate, Regular Schedule, Rejected License, Request, Road Transit-able Certificate, Route, Route Existing License, Route Supporting Documents, Schedule, Stakeholder, Stakeholder Supporting Documents, Stop, Subcontracting Contract, Tax Clearance Evidence, Transport License Service, Vehicle, Vehicle Existing License, Vehicle Inspection Certificate, Vehicle Insurance, Vehicle Supporting Documents Instances: none.
Source of knowledge	Guidelines and forms from Portugals transport related licensing services [81–83] Guidelines and forms from Irelands transport related licensing Services [54, 84, 85]

and edges between concepts to express their relationships – this constitutes a graphical representation of the ontology. The edges can be annotated with the kind of relation they represent – i.e. “is-a”, “instance-of”, “has-decomposition”, and “has-parts”; and the cardinality at both ends. The predefined relations and their notation can be seen in Figure 8.5 – relations read from left to right.

Each time a new kind of relationship is used it is necessary to define it in a supplementary table. For each relation, the table defines: name, reversed name (enabling to reading relationships both ways), purpose of the relation, the structure the relationship establishes on instances of the concepts, and properties of the relationship. Table 8.3 defines the new relations identified for the proposed ontology, following REFSENO methodology.

Table 8.2: Glossary of concepts

Name	Description
Application processing criteria	It specifies a set of criteria for modelling the application processing workflow.
Approved license	The outcome of an accepted license application.
License application	It represents all relevant information submitted in request of a license.
License application supporting document	Documentation that can be requested by the corresponding authorities to complete a valid application.
License decision	It represents the outcome of a license application.
License for passenger transport	A license that enables the holder to provide a public bus passenger transport service across specified pick up/set down points following a predefined schedule and fare scheme.
License for transport operator	A license that enables the holder to operate hire and reward passenger transport services.
License for vehicle	A license that enables a vehicle to be used for transporting passenger for hire and reward.
Life cycle stage per license	It defines possible status of the application, such as request, renew, amend, cancel, transfer, and revoke.
Rejected license	The outcome of a rejected license application.
Stakeholder	It represents a party involved in the process of requesting a license.
Stakeholder supporting document	Stakeholder's official documentation that can be requested by the corresponding authorities to make a valid application.
Transport license service	A service providing the necessary functionality for applying, processing, and issuing a particular type of transport license.

A graphical representation of the ontology for transport licensing services showing each concept and their relationships is depicted in Figure 8.6. The essence of the concepts and the relationships defined in the ontology are summarized below.

A Government Authority can provide various Transport Licensing Services. Each license service corresponds to one type of license (for example but not limited to, Passenger Transport, Transport Operator, and Vehicle) and provides functionality to one or more types of applications (Life Cycle Stage per License) for that type of license – e.g., request, renew, amend, cancel, etc. Each type of application for a particular license implements: 1) eligibility criteria that will support authorities in deciding whether to grant the license or not – e.g., suitability of applicant, interference with other granted licenses, etc. and 2) application processing criteria that defines procedural require-

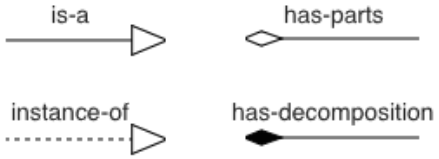


Figure 8.5: Predefine relation types

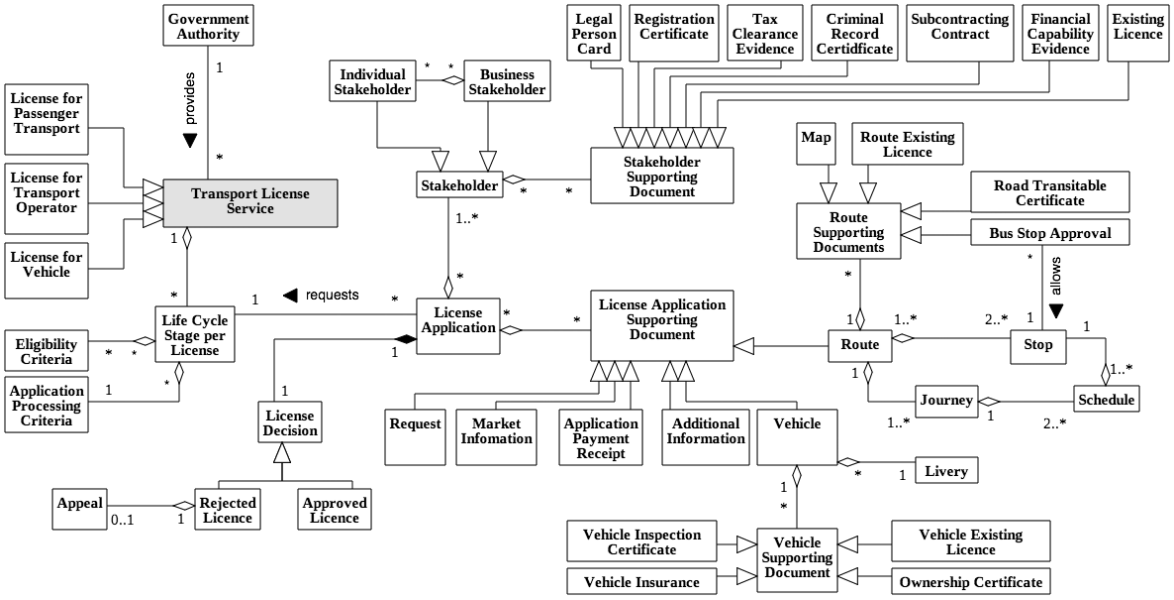


Figure 8.6: Transport licensing services ontology

ments for authorities when processing the applications and procedural requirements for applicants when submitting applications – e.g., deadlines for processing applications, whether resubmission of incomplete applications are allowed, if a fee is required, etc.

Each License Application involves various stakeholders, either individuals or businesses, such as the applicant (mandatory), members of the business in the case the applicant is a business, and subcontractors if the applicant intends to subcontract part of the future license obligations to other stakeholder. A license application may require various supporting documents for each stakeholder. The type of documents required will vary on the type of license, the type of the application and the actual implementation of the licensing services. In the proposed ontology, we define typical documents requested from stakeholders that were identified from the case studies: legal

Table 8.3: Custom relationships identified

Name	**Reverse Name**	**Purpose**	**Structure**	**Properties**
allows	allowed-by	The Bus Stop Approval document allows the pick up and set down of passengers in a Stop of a particular route. The same Stop is required to be approved for different routes.	DAG*	Transitivity
requests	requested-by	License Application requests a particular type of licensing service at a given Life Cycle Stage per License. Given the state, applications must conform to the applications pre-defined requirements for each license.	DAG*	Transitivity
provides	provided-by	A Government Authority provides Transport Licensing Services and is responsible for authorizing and regulating the issuing of licenses as well as ensuring accountability of the decision process.	DAG*	Transitivity

*DAG = Directed Acyclic Graph

person card and registration certificate (businesses only), tax clearance evidence, criminal record certificate, subcontracting contract, financial capability evidence, and other exiting licenses.

In addition, a license application requires different supporting documents that are related to the application itself and the type of application. As before, the required documents will vary depending on the type of license and type of application. Here we define typical documents required for the three types of licenses identified from the case studies: a formal request, proof of application payment, market information that can support the application, vehicle related information, route related information, and any additional information the applicant considers relevant.

Vehicle related information includes: information about the vehicles it self, intended livery for vehicles of a passenger transport service, and supporting document for the vehicles such certificates of insurances and inspections, proof of ownership, certificate, and previous licenses involving the vehicle, if any.

Information related to a route is typically required when applying for a passenger transport license. This includes: inherent information about the intended route to serve, information about bus stops, detailed schedule, and supporting documents such as a map of the city highlighting the route and bus stops, previous licenses of the route,

a certificate to attest that the route is transitable, and certificates of approval to pick up and set down passengers in each of the intended bus stops for the route.

Finally, a license application will result in a decision whether to accept or to reject the issuing of the license. In case the license application is rejected, the applicant may have the right to appeal such decision.

Concept attribute table

The fourth and fifth steps in the process model consist of identifying and defining terminal and non-terminal attributes for each of the concepts defined in the ontology. The methodology proposes a pre-defined table to capture such knowledge. The table is divided in two sections – concept related information, and attributes information. The former specifies the concept and its super-concept, if any. It is assumed that the concept inherits attributes from its super-concept. The latter specifies attribute information such as name, description, cardinality, type and whether it is mandatory or not. Both terminal and non-terminal attributes are defined in the concept table. However, for reasons of clarity and to respect the order in which the activities of each step are performed we present them here using two separate tables. We introduce these activities and present some results from the proposed ontology below. Attributes for each concept were extracted from guidelines and application forms from both case studies.

A terminal concept attribute serves to model how software engineering entities are specified for storage and retrieval. It can be seen as a property or a data element of a concept. Table 8.4illustrates various concept attributes tables for some of the main concepts identified in the ontology.

Table 8.4: Concept attribute table – terminal attributes

Concept	Transport License Service			
Super-Concept	-			
Name	**Description**	**Card**	**Type**	**Mand.**
license id	Identification code for the license	1	Text	Yes
license name	Name of the license	1	Text	Yes
Concept	License Cycle Stage per License			
Super-Concept	-			
Name	**Description**	**Card**	**Type**	**Mand.**
license duration	Time during which the license is granted	1	Integer	Yes
license duration unit	Unit of measure for the duration of the license	1	Date Unit	No

license fee	The fee to be paid by the applicant for issuing the license	1	Integer	Yes
application fee	The fee to be paid by the applicant for particular license life cycle	1	Integer	Yes
processing time	Indicative processing time of an application	1	Integer	Yes
processing time unit	Unit of measure for the license application processing time	1	Date Unit	No
license life cycle	A particular license life cycle stage that is available for a license service	1	License Life Cycle	Yes

Concept	Stakeholder
Super-Concept	-

Name	**Description**	**Card**	**Type**	**Mand.**
id	Identification code for a stakeholder	1	Integer	Yes
name	Stakeholders name – first, middle and last name of a person in the case of individuals, or business name in the case of businesses.	1	Text	Yes
address	Stakeholders primary address	1	Text	Yes
phone	Stakeholders phone number	1	Text	Yes
e-mail	Stakeholders e-mail address	1	Text	Yes
city	City of the stakeholders address	1	Text	Yes
zip-code	Zip-code of the stakeholders address	1	Text	Yes
role	Role of the stakeholder within the application and licensing process	1	Stakeholder Role	Yes

Concept	Business Stakeholder
Super-Concept	Stakeholder

Name	**Description**	**Card**	**Type**	**Mand.**
legal number	Number of the legal person	1	Text	Yes
business type	Type of business	1	Business Type	Yes

Card: Cardinality; *: many; 1:one; Mand.: Mandatory

Each time a new type of terminal attribute is identified, it shall be defined in a supplementary table. REFSENO contains some predefined types including: Boolean, Text, Integer, Date, Symbol (symbols ordered alphabetically), and OrderedSymbol (symbols ordered from lowest to highest). For simplicity, we consider the type "Attachment" (attribute of Stakeholder Supporting Document) as a predefine type. This type represents an attached digital file. Table 8.5 shows the types definitions for each new attribute type identified in Table 8.4. Each type definition includes the name of the type, the super-type, and the range of possible values for attributes of this type.

The legend DYNAMIC following the range definition informs that the range of possible values can be extended.

If the types table includes declarations of symbol types it is necessary to define a glossary of symbols including a narrative definition for each possible value. Table 8.6 shows the symbol definition for some of the symbols types identified.

Table 8.5: Types

Name	**Super-Type**	**Value Range**
Date Unit	OrderedSymbol	"Day", "Week", "Month", "Year"
License Life Cycle	OrderedSymbol	"Request", "Renewal", "Transference", "Amendment", "Cancelation", "Revocation"
Application Life Cycle	OrderedSymbol	"Submitted", "Processing", "Rejected", "Accepted" DYNAMIC
Payment Method	Symbol	"Card", "Cash", "Cheque", "Postal Order", DYNAMIC
Stakeholder Role	Symbol	"Applicant", "Business Member", "Subcontractor", DYNAMIC
Business Type	Symbol	"Company","Cooperative", "Partnership" "Sole Trader", DYNAMIC

Similarly, we identify non-terminal attributes. A non-terminal attribute models how a particular software engineering entity is related to other software engineering entities. It can be seen as an association to other non-terminal concept. Non-terminal attributes of the predefined kind "is-a" are not represented explicitly in the table since such relationship is represented through the declaration of the super-concept. Following the table structure introduced in the previous section, Table 8.7 illustrates the concept attribute tables with non-terminal attributes for the concepts defined in Table 8.4.

Table 8.7: Concept attribute table – non-terminal attributes

Concept	Transport License Service			
Super-Concept	-			
Name	**Description**	**Card**	**Type**	**Mand.**
life cycle	Stages in the license life cycle that the license service supports and provides functionality for	*	has-parts[Life Cycle Stage per License].[license service] * Yes	

responsible agency	Government agency responsible for the provision of the licensing service	1	part-of[Government Agency].[licensing services]	Yes

Concept	License Cycle Stage per License			
Super-Concept	-			
Name	**Description**	**Card**	**Type**	**Mand.**
license service	A particular type of transport license service for the license life cycle stage available	1	part-of[Transport License Service].[life cycle]	Yes
eligibility criteria	Eligibility criteria associated with a particular life cycle stage of a transport license service to support the decision-making when processing an application	*	has-parts[Eligibility Criteria].[license types]	Yes
application processing criteria	Application processing criteria to be considered when implementing the transport license application service	1	has-parts[Application Processing Criteria].[license types]	No
applications	Applications made to request this particular stage and license type	*	requested-by[License Application].[application type]	Yes

Concept	Stakeholder			
Super-Concept	-			
Name	**Description**	**Card**	**Type**	**Mand.**
supporting documents	Required documents related to the stakeholder that support the application	*	has-parts[Stakeholder Supporting Document].[stakeholder]	Yes
license application	License applications in which the stakeholder is involved	*	part-of[License Application].[applications]	Yes

Concept	Business Stakeholder			
Super-Concept	Stakeholder			
Name	**Description**	**Card**	**Type**	**Mand.**
related stakeholders	Stakeholders related to the business and their position in or relation to the business	*	has-parts[Individual Stakeholder].[related business]	No

Table 8.6: Glossary of symbols

Name	Super-Type	Value Range
License Life Cycle	Request	Request for a new license
	Renewal	Request to renew an existing license
	Transference	Request to transfer an existing license from one individual or business to another
	Amendment	Request to make changes to some of the terms and conditions of an existing license
	Cancellation	Request to cancel the validity of an existing license
	Revocation	Request to withdraw an existing valid license
Stakeholder Role	Business Member	A person that is member of or related to a business stakeholder
	Subcontractor	A stakeholder that posses a required license and is subcontracted to perform the obligations related to the license
	Applicant	A stakeholder that is the main responsible for the application process and the beneficiary of the license if granted

Completeness check

The sixth step in the process model involves checking the completeness of all concept attribute tables. As defined in Table 8.1, the purpose of the proposed ontology is to provide common vocabulary for the modelled domain with the intention of facilitating the generation of families of transport licensing services. This implies that the ontology will potentially be used to instantiate licensing public bus passenger services in very different environments – different countries with different laws and regulations. Therefore, the approach is only to define most elemental attributes for each concept. Each instantiation of the ontology can later define additional attributes and even additional concepts. Thus, completeness check is performed considering only elemental attributes that will likely be present in every instantiation of the concept. Based on this, each concept attribute table is complete with respect to the small set of such attributes.

Instantiation

The final step in building an ontology using REFSENO involves defining the instances specified in the ontology definition table (Table 8.1). For each instance, the methodology proposes a table containing an instance identification name, the concept associated to the instance, and the values for each of the attributes defined in the concept attribute

table. However, this goes out of the scope of the intended use.

8.3.3 Discussion

An advantage of building ontologies with REFSENO is that by construction it ensures: 1) completeness in the sense that all relevant knowledge to instantiate a knowledge base is defined; and 2) consistency in the sense that some consistency criteria have to be fulfilled during the construction such as: a) no concept, types, instances or attributes of a same concept have the same name, b) graphical representation of the non-terminal attributes and their relationships must match the tabular representation, etc. In addition, the table structure used to defined the ontology is easy to understand by domain experts.

The main aim of the proposed ontology is to facilitate the definition of generic models to support the definition of a family of software applications for licensing public transport services adopting SPL engineering methods and tools. However, by defining a common vocabulary, the ontology can serve other purposes: 1) facilitating the transition from paper-based delivery channels to electronic ones; 2) facilitating the integration of different licensing systems, and 3) improving government interoperability. The last two are important because they facilitate information sharing between agencies enabling the delivery of one-stop, seamless services, and the implementation of the "only-once" principle for reducing administrative burden [66].

As a limitation of the ontology, we highlight that the ontology itself does not define which supporting documents correspond to which type of license application.

8.4 Software Engineering

The software engineering component deals with the rapid modelling of digital public services in a given family and the automatic generation of software applications that support the delivery of such services, implementing the requirements specified by the models. In particular, it relies on component-based and software product line techniques. The current components of the virtual factory supporting this stage focus on the rapid modelling of behavioural aspects of a family of services, and comprise the following elements:

a) A *specification language* for service modelling and assembly by feature composition;

b) A *proof-of-concept prototype* to specify, compose, visualize, and translate the relevant models to other well-known formalisms;

c) A *verification engine* to check whether properties documenting the family are satisfied by the models.

In particular, the formalism used to define the behavioural models is IFTA as introduced in Chapter 5; while for the verification engine c) we use of-the-shelf tools, particularly the UPPAAL real time model checker, to which we translate IFTA models to verify temporal properties of the services in the family.

To populate the virtual factory and to illustrate its usage, we model some behavioural aspects of the identified family, providing in particular:

– A *feature model* specifying domain variability in terms of common and optional functionality present in the services of the selected family

– *Behavioural models* characterising features representing functionality of the domain and *Temporal properties* of such models

8.4.1 Case Study

This section presents the behavioural characteristics identified from the analysis of the family of services studied. In particular, we use IFTA to formally specify the behaviour. It is worth to mention that in both cases, Ireland and Portugal, the licensing services were being delivered by paper-based solutions. In this sense, some functionality described here, such as the support for online payments, was not present in the original case studies.

Feature model

We propose a feature model, illustrated in Figure 8.7, to express the valid combination of features present in the family of licensing public transport services. It follows directly from the identification of the structural (Section 8.3) and behavioural aspects (Section 8.4.1) of the case study.

The root feature represents the family of licenses. The top sub-features – license, documents, and license life cycle correspond to features identified from the structural analysis of the case study and coming from the respective elements of the ontology. Intuitively, only one type of license can be derived at a time (alternative features): transport operator, passenger transport, or passenger vehicle. Each license must have, at least (mandatory features), documents associated to the stakeholder and the application. The specific documents depend on each particular context (optional features), i.e. specific local government. A license service must support at least the request of a new license, but it can optionally provide functionality to support other requests in the

license life cycle. The last sub-feature of the root represents the support for business processes identified from the case study. All licenses provide support for three main process – submission of the application, processing of the application, and issuing of the license. In addition, some submissions require payments, which can support payments by credit card or by PayPal; and resubmitting documents in case some documents are missing. In addition, some licenses can support an appeal on a rejected license.

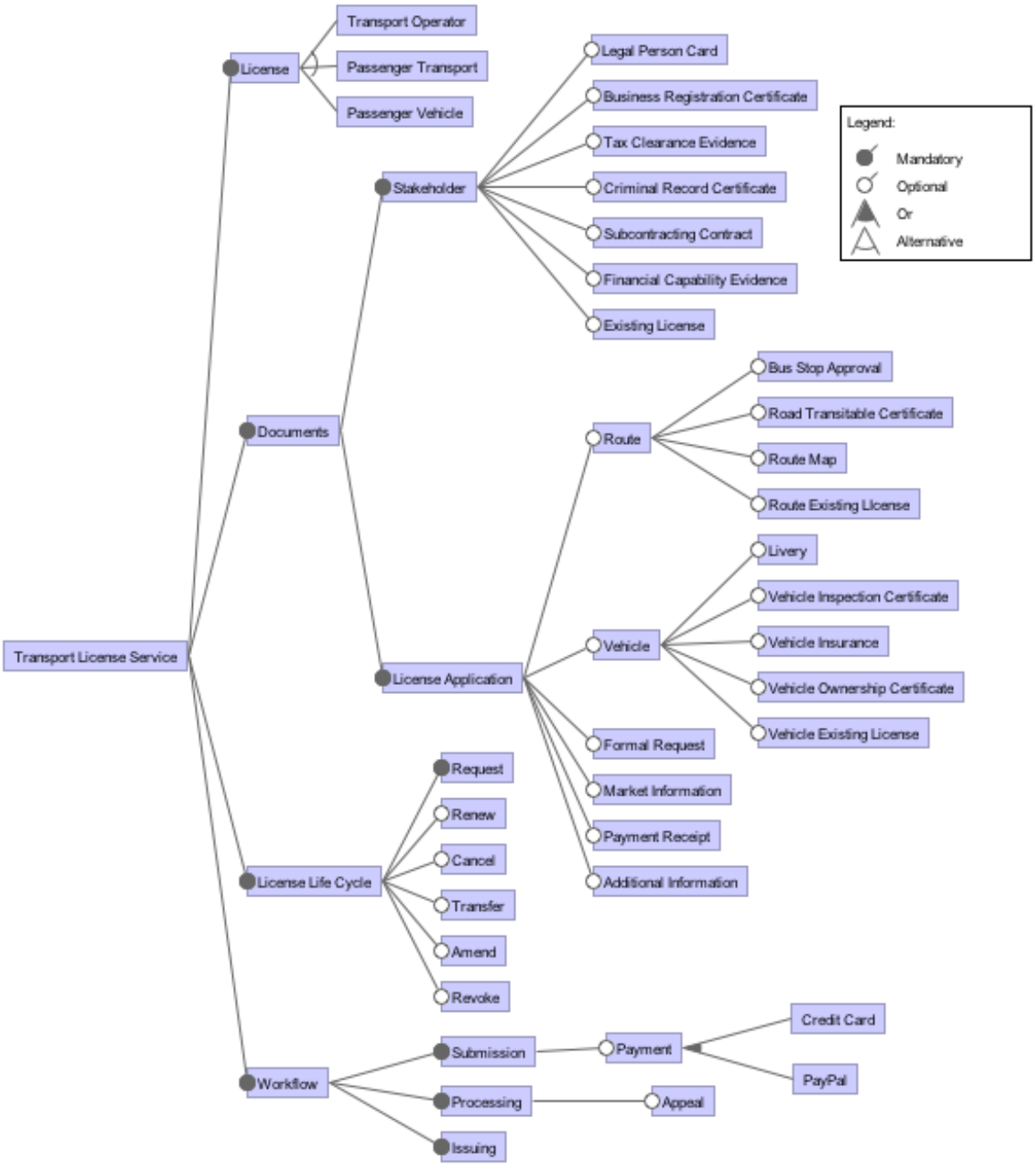


Figure 8.7: Proposed feature model for the family of public transport licensing services

Behavioural models

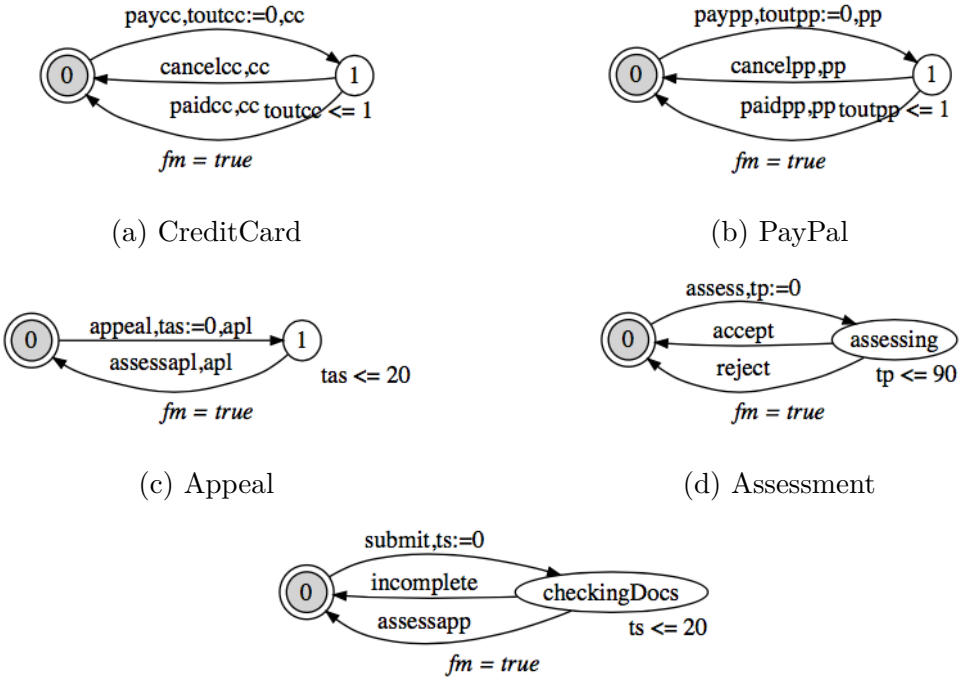
We exemplify the use of IFTA by producing models representing functionality associated to the features under the *workflow* feature in Figure 8.7.

The behaviour of these features, as identified from the analysis of the case studies, is as described below. All services in the family share a common business process: 1) submitting licensing requests (*submission*), 2) assessing requests (*processing*), and 3) issuing the corresponding decision (*issuing*). Some licensing services, in addition, support an online payment before submitting a request (*pa*), and appeals on rejected requests (*apl*). Furthermore, services that support online payments can support credit card payments (*cc*), PayPal payments (*pp*), or both. For simplicity, given that features *submission*, *processing* and *issuing* are mandatory, we avoid representing this feature and simply assign the feature expression $\top$ to the corresponding transitions.

Functionality is divided into components and provided as follows. The IFTA for each component can be visualized in Figures 8.8 and 8.9, while the architectural view of the composed family can bee seen in Figure 8.10 These figures have been automatically generated by the proof-of-concept prototype that we briefly explain in Section 8.4.2. We divided the processing stage into two components to simplify the design – one that deals with the completeness of the documentation, and another to assess the request. For simplicity, we use the same action name in two different automata to indicate a pair of actions to be linked.

Submission – the component models licenses requests. An applicant must submit the corresponding documentation (*subdocs*), pay for the application (*payapp*) if *pa* is present, and confirm the submission (*submit*). If the submission is accepted (*accept*) or considered incomplete (*incomplete*), then the application is closed, i.e. the request is no longer active and the applicant can start a new application. If it is rejected (*reject*) and it is not possible to appeal ($\neg$*apl*), the application is closed, otherwise a clock (*tapl*) is reset to track the time the applicant has to appeal. In particular, the applicant has up to 31 days to submit an appeal on the decision ($Inv_{App}(\ell_5)$), otherwise the application is cancelled (*cancelapp*) and closed. If an appeal is submitted (*appeal*), it can be rejected or accepted, and then the application is closed.

Credit Card – the component models payments through credit cards. If a user requests to pay by credit card (*paycc*), a clock is reset to track payment elapsed time (*topp*), after which the user has less than 1 day ($Inv_{CC}(\ell_1)$) to enter the details and proceed with the payment which can result in success (*paidcc*) or cancellation (*cancelcc*).



(a) CreditCard

(b) PayPal

(c) Appeal

(d) Assessment

(e) Preassessment

Figure 8.8: IFTA models for the Preassessment, Assessment, Appeal, Credit Card, and Paypal components.

PayPal – the component models payments through Paypal accounts. If a user requests to pay by Paypal (*paypp*), a clock is reset to track payment elapsed time (*tocc*), after which the user has less than 1 day ($Inv_{PP}(\ell_1)$) to enter login and proceed with the payment which can result in success (*paidpp*) or cancellation (*cancelpp*).

Appeal – the component models the process of handling appeal requests. When an appeal on a decision (*appeal*) is received, a clock is reset to track the appeal submission elapsed time (*tas*), after which the corresponding authority has up to 20 days ($Inv_{Appeal}(\ell_1)$) to start assessing the request (*assessapl*).

Preassessment – the component models the process of checking if a request contains all required documentation. When a submission is received (*submit*), a clock is reset to track the submission elapsed time (*ts*), after which the corresponding authority has up to 20 days ($Inv_{Preassesment}(\ell_1)$) to check if all required documents

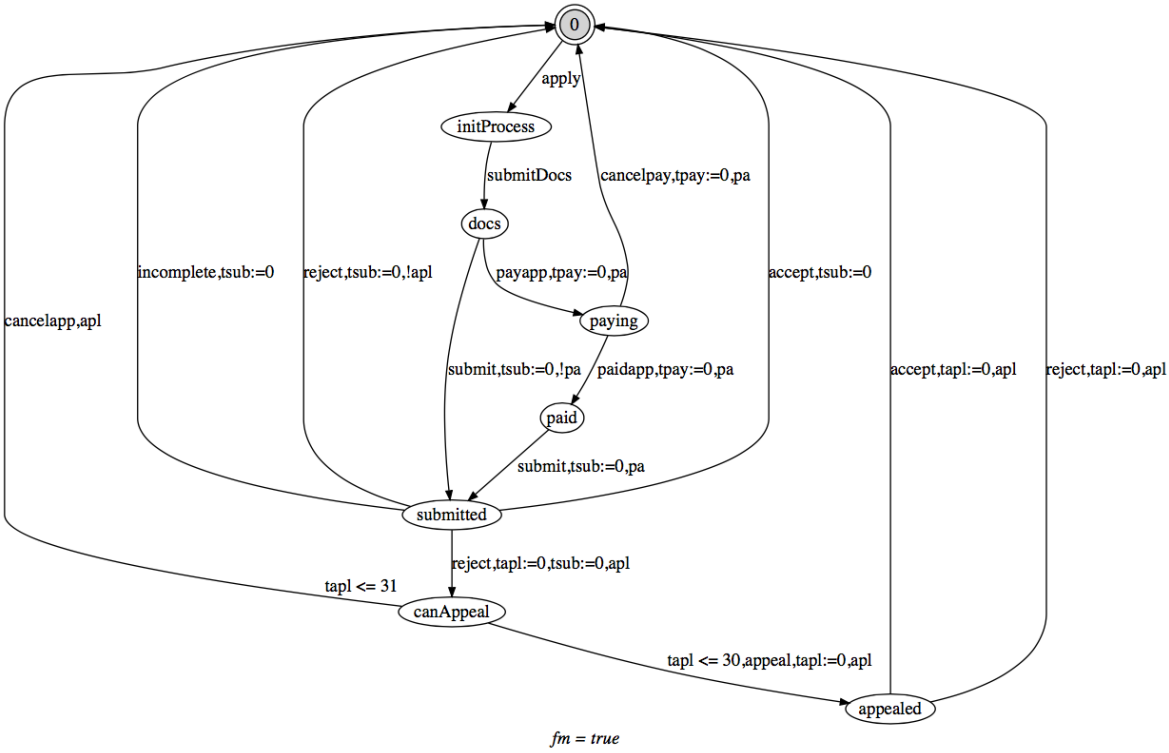


Figure 8.9: IFTA models for the Submission component.

have been submitted and notify of its incompleteness (*incomplete*) or proceed to the assessment of the request (*assessapp*).

Assessment – Models the process of analysing a request. When a request is ready to be assessed (*assess*), a clock is reset to track the processing elapsed time (*tp*), after which the corresponding authority has up to 90 days to make a decision of weather accept (*accept*) or reject (*reject*) the request.

We use a set of $\mathcal{R}$eo connectors in order to orchestrate the way these components interact as discussed in Chapter 7. In this case, it was necessary to use the relaxed approach to model each protocol.

The final model can be seen in Figure 8.10. For simplicity, we omit the feature expressions associated to ports and the resulting feature model. Broadly, we can identify three main components in this figure: (1) Application of requests (*Submission*), (2) Processing of requests (right of *Submission*), and (3) Payment of requests (below of *Submission*). The functionality of the new components is given as follows.

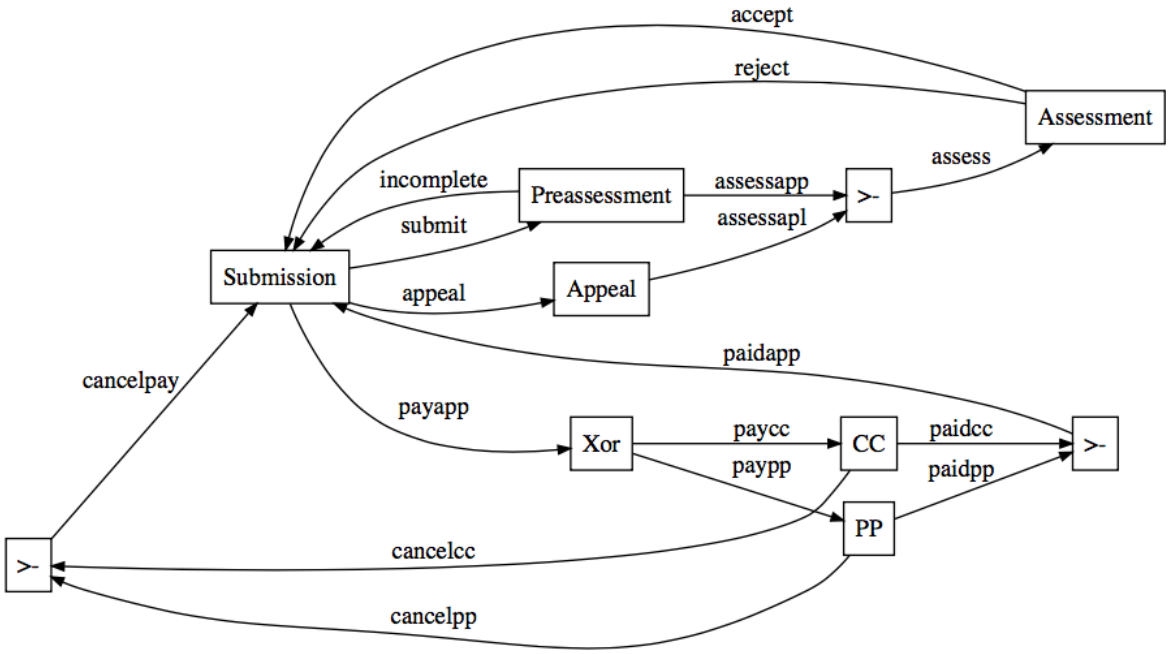


Figure 8.10: Architectural view of the composed family of licensing services

Payment – Models the orchestration of payment requests based on the availability of payment methods. It is composed by components *CC*, *PP*, a *router* and two *merger* connectors. If a request for payment is received (*payapp*) a *router* enables to choose between pay by Credit Card or Paypal (*paypp* or *paycc*). A *merger* synchronizes the successful response (*paidpp* or *paidcc*), while other *merger* synchronizes the cancellation response (*cancelpp* or *cancelcc*) from either *CC* or *PP*. In addition to the feature model generated by the composition of these IFTA, the payment component imposes the additional restriction that payment is supported by the system, if and only if, Credit card or Paypal payment are supported ($pa \leftrightarrow cc \vee pp$).

Processing – Models the orchestration of processing licenses requests and appeals on decisions (when *apl* is present). It is composed by components *Appeal*, *Preassess*, *Assess*, and a *merger* connector that synchronizes assessment request from either *Appeal* or *Preassess* (*assessapl* or *assessapp*) with *Assess* (*assess*).

As composition of IFTA combines the variability models of each individual automata and imposes new restrictions based on the interfaces connected, the resulting model will adapt automatically based on the feature selection. For example, if *pp* is

not present, the router in Figure 8.10 will behave as a *Sync* connector, synchronizing only actions *payapp* and *paycc*. Similarly, both merger that synchronize the outputs of *PP*, will behave as a *Sync* that synchronizes *cancelcc* with *cancelpay*, and *paidcc* with *paidapp*. In Section 7.2 we illustrated using a network of IFTA specifying a complex connector how the prototype developed facilitates the visualization of these changes based on the possible feature selections.

By specifying the previous components using the prototype described in the next section, we can translate them to Uppaal as a network of TA, or as an unique TA resulted from using IFTA composition, and verify properties such as:

- **Deadlock free** – `A[] not deadlock`;
- **Liveness** – a submition will eventually result in answer (`App.`ℓ_4 `-> App.`ℓ_0), an appeal will eventually result in answer (`App.`ℓ_6 `-> App.`ℓ_0), etc.; and
- **Safety** – it can not take longer than 110 days to process a submission (`A[] App.`ℓ_4 `imply App.tsub <=110`), it can not take longer than a day to pay a request after the payment was initiated (`A[] App.`ℓ_2 `imply App.tpay <=1`), etc.

8.4.2 Prototype

This section presents the proof-of-concept prototype implemented in the context of this thesis, and discusses some implementation decisions. Further documentation and discussion can be found in [37].

The prototype was developed in Scala[1] and consists of a small Domain Specific Language (DSL) to support specification of IFTA and networks of IFTA (NIFTA), and some operations over them. By a network of any kind of automata we understand a set of automata composed in parallel (||) and synchronized over a set of shared actions.

Scala DSL for IFTA

Some of the main features supported by the DSL include: 1) specification of (N)IFTA, 2) composition, product and synchronization over IFTA, 3) conversion of (N)IFTA to (networks of) FTA (NFTA) with committed states (CS), 4) conversion of (N)IFTA and NFTA to DOT[2] graph description language, 5) conversion of (N)IFTA to an interactive `.html` file using the Vis.js[3] visualization library, and 6) conversion of NFTA to Uppaal

[1] https://github.com/haslab/ifta
[2] http://www.graphviz.org/about/
[3] http://visjs.org

networks of TA (NTA) with features. We informally explain the DSL and some of the operations through a simple example.

8.4.1 Example. The payment network described in Section 8.4.1 can be specified using the DSL as shown in Listing 8.1. A new automaton is created with the constructor `newifta`, which builds an empty IFTA. New transitions are added through the operator `++`. In the case of the credit card IFTA, three transitions are specified between parenthesis, followed by the declaration of the ports. Each transition in the example specifies:

- the origin and destination location indicated by natural numbers and by the linking operator between them `->`;
- the actions labelling the transition, using the operator `by` followed by the set of actions encoded as a string where actions are separated by a comma;
- the associated feature expression, using the operator `when`, followed by the feature expression, where features are expressed as strings and the logical operators are encoded as usual, `&&`, `||`, `->`, `not`, and `<->`; and
- the set of clock to reset, using the operator `reset`, preceding each clock to reset, encoded as a string, and any clock constraint if it were the case.

Each port, encoded as a string, is preceded by the operator `get`, if it is an input port, and `pub` if an output port. The feature model is specified by a feature expression using the operator `when` as before. In this case the feature model is not specified, thus it is automatically assumed that the feature model is $\top$. A net can be created by composing automata in parallel as in the case of `paymentNet`. The DSL provides a set of $\mathcal{R}$eo connector constructs, such as the `router` and `merger` used in the example. Finally, the net is converted to a network of Uppaal TA, `toUppaal`, and stored in a given XML file. ◁

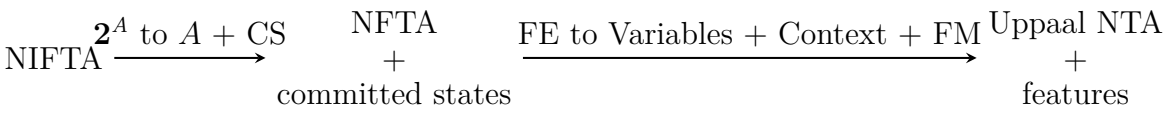
The full list of the functionality provided and other examples, including the case study presented in Section 8.4.1 can be found in the implementation repository.

Conversion to other formalisms

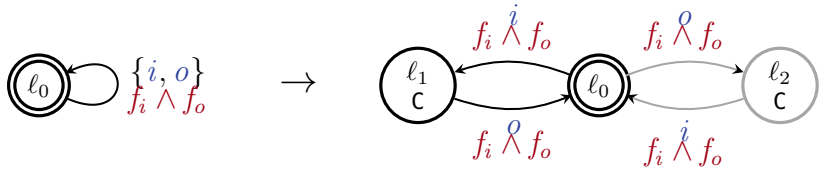
A network of IFTA can be step-wisely converted into a network of FTA with committed states, which in turn can be converted into a network of Uppaal TA, as follows.

```
val creditcard = newifta ++ (
    0 --> 1 by "paycc" when "cc" reset "toutcc",
    1 --> 0 by "cancelcc" when "cc",
    1 --> 0 by "paidcc" when "cc"
    ) startWith 0 get "paycc" pub "cancelcc,paidcc" inv(1,"toutcc"<=1) name "CC"

val paypal = newifta ++ (
    0 --> 1 by "paypp" when "pp" reset "toutpp",
    1 --> 0 by "cancelpp" when "pp",
    1 --> 0 by "paidpp" when "pp"
    ) startWith 0 get "paypp" pub "cancelpp,paidpp" inv(1,"toutpp"<=1) name "PP"

val paymentNet = (
    router("payapp", "paycc", "paypp") ||
    paypal || creditcard ||
    merger("cancelcc", "cancelpp", "cancelpay") ||
    merger("paidcc", "paidpp", "paidapp")) when "pa" <-> ("pp" || "cc")

// To Uppaal network of TA
toUppaal(paymentNet,"mynetwork.xml")
```

Listing 8.1: Example specification of a network of IFTA using the prototype DSL.



1) NIFTA to NFTA. Informally, this is achieved by converting each multi-action transition to a set of transitions with single actions that must execute atomically. The atomicity is achieved through committed states between them. In addition, the new set of transitions should support all possible combinations of execution order in the original multi-action transition. However, in practice this can quickly lead to a state explosion. To reduce this problem, we allow only combinations were the execution flows from inputs to outputs. For example, the following IFTA on the left is converted into the FTA on the right. Ideally, i and o should be enabled at ℓ_0, however, we only model the combination were i executes followed by o (dark arrows). The other possibility when o executes first (light grey) is not created. It is worth to mention that the word *first* refers here to the syntactic order, since semantically, due to the committed locations, the sequence of transitions is done atomically, i.e. time does not pass.

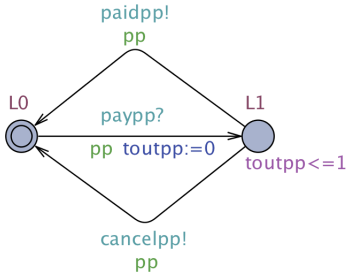


2) NFTA to Uppaal NTA. To create a network ready for simulation and verification in Uppaal, this activity involves three steps. Firstly, takes the NFTA obtained in the previous step and creates an Uppaal TA per each FTA in the network, where features are encoded as boolean variables, and transition feature expressions as a logic guard over Boolean variables. Secondly, the *feature model* of the network is solved using a SAT solver to find the set of all valid feature selections (or products). This set is encoded as a TA with an initial committed location and with an outgoing transition to a new location for each element in the set. Each transition represents a valid selection of features by initializing the corresponding set of variable representing those features. The initial committed state of the feature model that ensures a feature selection is made before any other transition is taken. Thirdly, a TA is created to represent the *context* of the network, which corresponds to the interface of the network. The context is represented as TA with an unique state and a loop transition for each action of the context. Figure 8.11 illustrates how the IFTA of the PayPal component, and the feature model of the payment net (Listing 8.1) are translated into Uppaal as TA. The composed feature model allows four products, from top to bottom: 1) feature *cc*, 2) features *pp* and *cc*, 3) feature *pp*, and 4) none. The additional features modelled with a prefix v_, represent generic features associated to the connectors.

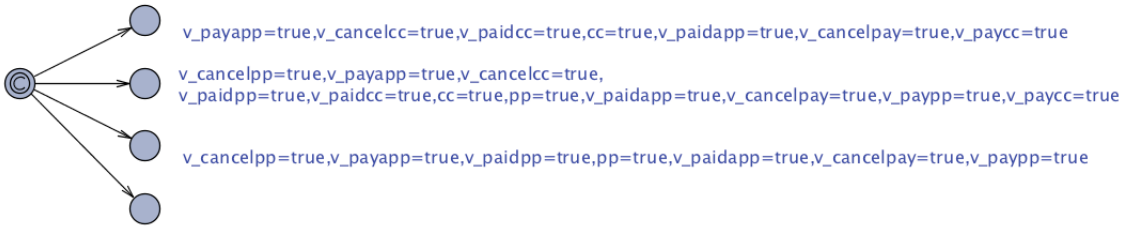
A main issue when translating networks of IFTA to networks of Uppaal TA are sequences of committed states. Because of how Uppaal deals with such sequences, it can lead to a deadlock when verifying properties. When the model checker *sees* that the first transition in such type of sequence is enabled, it executes the transition, moving to the next committed state. If there are no enabled transitions from that state, or another committed state, then the system is in a deadlock. This is a problem inherent to Uppaal rather than a problem of the model.

Another issues when translating IFTA to FTA with committed states, is that the complexity of the model grows quickly. For example, the IFTA of a simple replicator with 3 output ports consists of a location and seven transitions, while its corresponding FTA consists of 23 locations and 38 transitions. Without any support for composing variable connectors, modelling all possible cases is error prone and it quickly becomes unmanageable.

This simplicity in design achieved through multi-action transitions leads to a more efficient approach to translate IFTA to Uppaal TA in particular by using the composition of IFTA. The IFTA resulting from composing a network of IFTA, can be simply



(a) PayPal as Uppaal TA



(b) Feature model as Uppaal TA

Figure 8.11: Example of an Uppaal TA consisting of the *PayPal* component (Figure 8.8b), and the feature model of the payment net specified in Listing 8.1.

converted to an FTA by *flattening* the set of actions in to a single action, and later into an Uppaal TA, avoiding the use of committed states.

In addition to this translations, it is possible to convert the networks of IFTA and IFTA to an HTML file with JavaScript code. In this case, one may interactively select features and see how the selection affects the models, as it was illustrated in Section 7.2.

8.5 Discussion

The aim of this chapter was to introduce the virtual factory approach for digital public service development. This is a conceptual framework for the rapid development of digital public services for a given smart city domain. The approach was illustrated by taking the smart mobility dimension as a case study.

In particular, we proposed a taxonomy of smart mobility services which deals with the understanding of the domain, the identification of types of services to be delivered, technologies used, public value delivered by the services, and stakeholders to whom they can be delivered. Thus, it provides a tool for policy makers to plan and design smart mobility services.

Through planning, it is possible to identify a concrete family of services. Firstly,

the virtual factory fixes the vocabulary of the domain, actors, documents, processes, involved in the delivery of the family, through ontologies to structure information. From the ontology it follows immediately the features of the domain, which facilitates the creation of a feature model. Secondly, the virtual factory proposes to use the vocabulary identified to produce behavioural models and associate them to such features, making it possible to verify properties of the services.

By using techniques from component-based and SPL development it is possible to quickly model the behaviour of the entire family in a compositional way, and to derive models of concrete services by selecting a desired set of features.

The approach suggested presents some advantages over other silo-based approaches that can contribute to address some of the challenges discussed in Section 2.2.2. In particular:

- *rapid development* – by identifying families of services in a proactive manner through the use of planning tools, and develop them using SPL techniques;
- *service and business process integration* – by using ontologies to standardize the vocabulary of the domain it can contribute to sharing information among different government levels. In addition, the development of digital public services in a SPL brings the benefits of mass production and customization, contributing to the easy adoption of standardized but customized solutions by different local governments, facilitating sharing information across different agencies and government levels;
- *conformance with laws and regulations* – by using formal methods to model services en verifying them against expected properties;
- *usability* – using an SPL approach can increase the quality of the delivered products. Intuitively, the SPL enables to deliver products that can be quickly adopted by many clients – more government agencies using the services in a family would lead to more feedback to improve the services which will increase the quality of the services.
- *development costs* – by using and SPL approach; and
- *matching citizen needs* – the use of a taxonomy to plan the delivery of services contributes to move to qualitative approach to deliver DPS to citizens, since governments can identify first public value that needs to be delivered, instead of delivering services in a quantitative approach.

We envision a fourth component of the virtual factory, an implementation stage, to automate the development of software applications implementing the structural and

behavioural models specified in the domain and software engineering stages. A feasible approach would be to define a formal model to link the structural elements of the domain, and the behavioural and variability models with concrete code implementing the specification provided by such models.

Finally, although not provided in this thesis, we envision a formal mechanism to link each feature of the feature diagram with the structural elements of the domain, similar to the way IFTA uses feature expressions to annotate behavioural models. In addition, such a mechanism should link each structural element of the domain with corresponding behavioural models. For example, in the case of the documents, the typical behaviour to be supported is the CRUD cycle: creating, reading, updating, and deleting documents.

Chapter 9

Conclusions and Future Work

The aim of this research work was to provide a conceptual framework with tools for both, government officials and software developers, to rapidly plan and design integrated smart city DPS on a specific city domain. In particular, by relying on software product line techniques and formal methods to address concrete challenges in the development of DPS, such as rapid development of software applications supporting the delivery of DPS, service integration, reducing costs in service development, and ensuring conformance with laws and regulations. Further challenges were discussed in see Section 2.2.2.

For the research problem defined in the thesis, the following research questions where formulated:

RQ1) What kind of smart mobility services are delivered in the context of smart cities?

RQ2) How are such services delivered?

RQ3) What kind of public value is delivered by smart mobility services and to whom?

RQ4) Which modelling technologies are suitable for specifying common features of a family of digital public services delivered by local governments in the context of smart cities?

RQ5) Based on such modelling techniques, how to provide a domain-specific framework, including modelling tools that can automatically generate behavioural models for the members of the identified family of digital public services?

RQ6) Which family of digital public services in the context of smart mobility is amendable to be deliver through common business processes by local governments in different contextual conditions?

In order to understand the background of the domain we began by exploring DPS across the various dimensions of a smart city. However, we focused on smart mobility, due to the positive impact theses services have in the development of smart cities, as discussed in Section 2.4. We conducted a state of the art review to address research questions RQ1, RQ2 and RQ3, and consolidated the findings in a taxonomy of smart mobility service. The taxonomy serves as a tool for government practitioners, such as policy makers and government officials involved in the strategic planning of smart mobility initiatives, by identifying types of services to be delivered, stakeholders to whom deliver such services, and public value delivered, facilitating the definition of business cases for developing initiatives, as well as for prioritizing what services should be delivered. In addition, the taxonomy provides a common vocabulary to describe, discuss and share information about smart mobility initiatives, which can contribute to the sharing of knowledge and good practices among different governments. Further usages of the taxonomy by different stakeholders was discussed in Section 8.2.

The taxonomy also contributed to address RQ6. Most of the services identified by the taxonomy were developed by non-government entities or co-created between government and non-government entities. In this sense, the role of government seems to move away from service provider, to become an enabler and regulator of such services. With this in mind, and the fact that governments must ensure the provision of public transport services as a basic service to city residents , we identified a family of licensing public transports services to serve as a concrete case study. Licenses help government to ensure the provision of services complies with the established regulations. In addition, they are provided by local governments worldwide, and as such, are widely used by governments with different levels of resources and legal backgrounds.

To understand how services in the identified family were being delivered, we studied licensing public bus passenger services in two countries, Ireland and Portugal. From the analysis we identified 1) licenses required for the provision of such services, 2) documentation required for the application of each license, 3) entities involved in the provision of the licensing services, and 4) application process activities. In order to fix a common vocabulary for the domain, in particular considering the structural properties, we proposed an ontology of licensing public transport services, modelling structural elements, their attributes, and relationships.

The main aim of the ontology was to facilitate the definition of generic structural models for the given family. The ontology will serve to relate the structural models with behavioural models and parametrize both by features. This will be further explore as future work. The development of the ontology contributed to a better understanding of the domain and due to the standardizing nature of ontologies, it serves other purposes as well, including: the automation of service features so to facilitate the transition from paper-based delivery channels to electronic ones and the integration of different

licensing systems; and improving government interoperability.

As mentioned, during the analysis of the family we identified the processes required to obtain and issue the licences. At this stage, it was evident that *time* was a recurrent requirement for the analysed services and that processes were distributed across various actors, which required service *coordination*. Thus, we studied SPL modelling formalisms that could deal with such requirements. We identified FTA as a promising formalism which included tools for verification. However, since FTA is a fine grain approach to model SPL, it could not deal with model composition. As discussed in Section 4.1.2, modelling complex SPLs requires modular and scalable modelling formalisms. In particular, the complex nature of DPS due to the different stakeholders and processes involved in their delivery, requires a specification formalism capable of simplifying their design, as well as, of delegating the modelling of different processes to the relevant stakeholders. The simplification can be achieved, for example, by dividing functionality into simpler modules. The findings of this stage contributed to answer the research question RQ4. Details of the proposed solution are explained below

We proposed a compositional formalism, IFTA, based on FTA. The formalism explicitly defines interfaces for the models, which restrict the way automata interact. This interfaces are variable, i.e. they are parametrized by features and may not be present in every product that can be derived from the SPL. The main contribution of IFTA is its compositionality, which makes possible the incremental design of SPLs. We argue that this is a truly compositional formalism in the sense that it composes families of systems, instead of products, as most approaches do in the literature (see Section 4.1.2). In addition, because composition composes variability as well, it is possible to verify if the set of products that can be derived from the composed system it is the intended. We implemented a proof-of-concept prototype to specify, compose, visualize, and translate IFTA models to other formalisms. The most relevant functionalities of the prototype are the translation of models into Uppaal networks of TA, which enables the verification of behavioural properties over the models; and the interactive visualization of IFTA and networks of IFTA, which enable to quickly see which are the valid set of products that can be derived from the models, and how each particular selection affects them. In particular, it is possible to visualize how feature selections affect transitions, in the case of IFTA models, and interfaces and connections between different components, in the case of networks of IFTA.

Towards addressing the issue of delegating the design and development of systems to various stakeholders, we proposed a refinement relation for IFTA. The relation serves to compare two IFTA – one representing an abstract model specifying a family of systems, and another representing a more detailed implementation of such specification; and it helps to determine if the implementation can replace the specification in every environment where the specification is used, to obtain a congruent system.

The are various notions of refinement that could have been considered as discussed in Section 6.2, each with its advantages and limitations. Here we opted for defining a relation that maintains the reactivity of the systems, i.e. it ensures that the implementation can react to the same inputs than the specification, and that it will not produce unexpected outputs. In this trade-off, the relation does not ensures that the implementation preserves safety properties of the specification. The relation is a *pre-order* and *compositional*. The latter allows decomposition of refinement proofs, improving efficiency in refinement checking. However, to simplify the definition, we separated the notion of variability refinement from behavioural refinement, which entails to conduct refinement proofs following a product-by-product approach, hindering efficiency. Thus, we proposed as well a variability-aware refinement relation that can be applied over the entire family.

Related to the modular and scalable approach to model SPLs, and the need to coordinate services with variable interfaces, we studied $\mathcal{R}$eo, an exogenous coordination language to orchestrate how components interact through their interfaces. In accordance with the principles of SPL, namely the development of reusable assets for systematic reuse in the development of products, we argue that exogenous coordination presents an opportunity for increasing the reusability of both, domain models, and models of the coordination protocols used to orchestrate them. Since in our case, components have variable interfaces, coordination protocols need to adapt to their presence or absence. We proposed to model $\mathcal{R}$eo connectors as IFTA, providing them with a variable semantics. The original intention was to enable protocols to automatically adapt their variability based on the variability of the components they interact. However, in practice, this is not a simple task, and it required to model connectors with different degrees of variability depending the case. In particular we proposed two different approaches to model $\mathcal{R}$eo, a conservative and a relaxed one, offering different degrees of variability. However, other approaches may be required. In practice, connectors from both approaches are combined to achieve an expected outcome. Nevertheless, using IFTA to model connectors simplifies significantly the design of coordination protocols. The prototype developed was of utmost help to visualize complex coordination protocols and understand which approach should be used for modelling each connector.

Finally, we introduced the concept of a virtual factory, which aims at facilitating the rapid development of models and software applications supporting the delivery of DPS. This contributed to address RQ5. The virtual factory seeks to shift the paradigm from silo-based development of services to a component-based development, by identifying families of services from an early stage and accompanying various stakeholders through the various stages in the development of DPS. The concept relies on three principles: 1) service family specific - the virtual factory is configurable for a given family, in the case of this thesis, for a family of licensing of public transport services;

2) scalable it can be extended with other components; and 3) generalizable it can be applicable to other families. In this thesis, we populate the virtual factory with components that contribute to the planning, domain engineering, and software engineering stages of DPS development of smart mobility services. In particular, the taxonomy contributed to the strategic planning of smart mobility initiatives; the ontology contributed to the domain engineering stage by providing structural models and fixing a common vocabulary for the concrete family; and the specification formalism, IFTA and related contributions, namely the refinement relation, models for $\mathcal{R}$eo, and the proof-of-concept prototype, contributed to the software engineering stage, providing mechanism to specify behavioural aspects of DPS and verify properties.

Future Work

The research conducted as part of this thesis enables to define several problems to be explored as future work, as discussed below.

The taxonomy of smart mobility services introduced in Section 3.3 can be extended and improved following guidelines and suggestions provided in Section 3.5.

The ontology introduced in Section 8.3 can be formalized using the W3C Web Ontology Language[1] (OWL) to verify properties such as consistency, or to infer new knowledge. In addition, as envisioned, it is necessary to define a mechanism to associate the ontology elements with variability, as well as with behavioural models related to such elements. A possible path is to define a meta-ontology comprising *products*, to which we associate *elements* which correspond to structural elements of the domain. Then, we can associate *behavioural models* to such elements, and finally, we associate *features* to elements and behavioural models. With the assistance of a feature model documenting valid feature combinations, the envision mechanism should enable selecting a product, for example a concrete public transport license like transport operator, and infer which features are available for such product. Finally, based on a concrete feature selection, to offer tools to derive the concrete structural and behavioural models for the selected features.

Regarding IFTA, there are many opportunities for exploration particularly concerning refinement. As discussed in Chapter 6 it is of interest to study other alternatives of behavioural and variability refinement. In particular, exploring how these new definitions affect the properties that we can expect from a refinement relation, mainly regarding compositionality and the properties that the implementation can preserve from the specification. In the case of variability refinement, one possibility to explore is to allow the feature model of the implementation to introduce new behaviour through new features, which would align better with the notion of behavioural refinement. In

[1] https://www.w3.org/standards/techs/owl

the case of behavioural refinement, simulation relations are usually used as refinement relations. Such relation preserves safety properties at the expense of allowing the implementation to react to less inputs. It would be interesting to explore if a new notion of refinement can be achieved for families of IFTA, to preserve both, the reactivity of the system and the safety properties satisfied from the specification. In addition, other ways of composing feature models can be explored when defining IFTA composition.

Another area of exploration for IFTA are $\mathcal{R}$eo models. As discussed, in practice different degrees of variability are required depending on the context. As future work, other approaches to model $\mathcal{R}$eo connectors can be explored to simplify even more the definition of these protocols taking away the burden from the user from selecting the right approach. A possible path can be to work on a constraint solver to automatically suggest variability restrictions in order to achieve a given desired outcome.

Finally, regarding the virtual factory, new components can be added such as: a Domain Specific Language for domain users (non-software engineers) to easily specify services; a mechanism for automatic code generation based on structural and behavioural models; a suitable logic for specifying and verifying structural properties of the domain; and its adaptation to other domains of a smart city and other families within the smart mobility domain.

Bibliography

[1] M. Acher, P. Collet, P. Lahire, and R. France. Composing feature models. In M. van den Brand, D. Gašević, and J. Gray, editors, *Software Language Engineering*, pages 62–81, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[2] A. Aldama-Nalda, H. Chourabi, T. a. Pardo, J. R. Gil-Garcia, S. Mellouli, H. J. Scholl, S. Alawadhi, T. Nam, and S. Walker. Smart cities and service integration initiatives in North American cities. In *Proceedings of the 13th Annual International Conference on Digital Government Research*, page 289, New York, New York, USA, 2012. ACM Press.

[3] L. Allulli, G. F. Italiano, and F. Santaroni. Exploiting GPS Data in Public Transport Journey Planners. In *13th International Symposium on Experimental Algorithms*, 2014.

[4] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994.

[5] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. *Alternating refinement relations*, pages 163–178. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[6] G. Anastasi, M. Antonelli, A. Bechini, S. Brienza, E. D'Andrea, D. De Guglielmo, P. Ducange, B. Lazzerini, F. Marcelloni, and A. Segatori. Urban and social sensing for sustainable mobility in smart cities. *2013 Sustainable Internet and ICT for Sustainability, SustainIT 2013*, 2013.

[7] S. Apel, D. Batory, C. Kstner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated, 2013.

[8] S. Apel, H. Speidel, P. Wendler, A. von Rhein, and D. Beyer. Detection of feature interactions using feature-aware verification. In *Proceedings of the 2011*

26th IEEE/ACM International Conference on Automated Software Engineering, ASE '11, pages 372–375, Washington, DC, USA, 2011. IEEE Computer Society.

[9] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, (3):329–366, 2004.

[10] F. Arbab. *Puff, The Magic Protocol*, pages 169–206. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[11] F. Arbab, C. Baier, F. de Boer, and J. Rutten. Models and temporal logical specifications for timed component connectors. *Software & Systems Modeling*, 6(1):59–82, Mar 2007.

[12] F. Arbab, N. Kokash, and S. Meng. *Towards Using Reo for Compliance-Aware Business Process Modeling*, pages 108–123. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[13] F. Arbab and J. J. M. M. Rutten. *A Coinductive Calculus of Component Connectors*, pages 34–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[14] R. Arnott and E. Inci. An integrated model of downtown parking and traffic congestion. *Journal of Urban Economics*, 60(3):418–442, 2006.

[15] C. Baier. Probabilistic models for reo connector circuits. *Journal of Universal Computer Science (J. UCS)*, 11(10):1718–1748, oct 2005.

[16] C. Baier, J.-P. Katoen, and K. G. Larsen. *Principles of model checking*. 2008.

[17] C. Baier, M. Sirjani, F. Arbab, and J. Rutten. Modeling component connectors in reo by constraint automata. *Sci. Comput. Program.*, 61(2):75–113, July 2006.

[18] K. D. Bailey. *Typologies and Taxonomies: An Introduction to Classification Techniques*. Number 07. 1994.

[19] M. Bakillah, S. H. L. Liang, and A. Zipf. Toward coupling sensor data and volunteered geographic information (VGI) with agent-based transport simulation in the context of smart cities. *Proceedings of the First ACM SIGSPATIAL Workshop on Sensor Web Enablement - SWE '12*, pages 17–23, 2012.

[20] F. Bannister and D. Wilson. O(Ver)-Government?: Emerging Technology, Citizen Autonomy and the Regulatory State. *Information Polity*, 16(1):63–79, Jan. 2011.

[21] R. Barrero, J. V. Mierlo, and X. Tackoen. Enhanced Energy Storage Systems for Improved On-Board Light Rail Vehicle Efficiency. *IEEE Vehicular Technology Magazine*, (September):26–36, 2008.

[22] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 30(6):355–371, 2004.

[23] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, Sept. 2010.

[24] C. Benevolo, R. P. Dameri, and B. D. Auria. Smart Mobility in Smart City - Action Taxonomy, ICT Intensity and Public Benefits. 11:13–29, 2016.

[25] J. Bengtsson and W. Yi. *Timed Automata: Semantics, Algorithms and Tools*, pages 87–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[26] J. Bertot, E. Estevez, and T. Janowski. Universal and contextualized public services: Digital public service innovation framework. *Government Information Quarterly*, 33(2):211–222, 2016.

[27] P. Blythe. RFID for road tolling, road-use pricing and vehicle access control. In *IEE Colloquium (Digest)*, number 123, pages 67–82, 1999.

[28] D. Bruno and H. Richmond. The Truth About Taxonomies. *Information Management Journal*, 37(2):44–53, 2003.

[29] C. Busold, A. Taha, C. Wachsmann, A. Dmitrienko, H. Seudié, M. Sobhani, and A.-R. Sadeghi. Smart keys for cyber-cars: secure smartphone-based NFC-enabled car immobilizer. *Proceedings of the third ACM conference on Data and application security and privacy*, pages 233–242, 2013.

[30] J. Candamo, M. Shreve, D. Goldgof, D. Sapper, and R. Kasturi. Understanding Transit Scenes: A Survey on Human Behavior-Recognition Algorithms. *IEEE Transactions on Intelligent Transportation Systems*, 11(1):206–224, mar 2010.

[31] E. M. Cepolina and A. Farina. A new shared vehicle system for urban areas. *Transportation Research Part C: Emerging Technologies*, 21(1):230–243, 2012.

[32] S. L. Cisco and W. K. Jackson. Creating Order out of Chaos with Taxonomies. *Information Management Journal*, 39(3):44–50, 2005.

[33] D. Clarke, D. Costa, and F. Arbab. Connector colouring i: Synchronisation and context dependency. *Science of Computer Programming*, 66(3):205 – 225, 2007. Special Issue on the 4th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA 05).

[34] A. Classen, Q. Boucher, and P. Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Sci. Comput. Program.*, 76(12):1130–1143, Dec. 2011.

[35] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay. Symbolic model checking of software product lines. *International Conference on Software Engineering, ICSE*, pages 321–330, 2011.

[36] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. Model checking lots of systems: efficient verification of temporal properties in software product lines. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 335–344. ACM, 2010.

[37] G. Cledou. A proof-of-concept prototype for ifta. Technical report, HASLab INESCTEC and University of Minho.

[38] G. Cledou. A virtual factory for smart city service integration. In *Proceedings of the 8th International Conference on Theory and Practice of Electronic Governance*, ICEGOV '14, pages 536–539, New York, NY, USA, 2014. ACM.

[39] G. Cledou and L. S. Barbosa. An ontology for licensing public transport services. Technical report, HASLab INESCTEC and University of Minho.

[40] G. Cledou and L. S. Barbosa. An ontology for licensing public transport services. In *Proceedings of the 9th International Conference on Theory and Practice of Electronic Governance*, ICEGOV '15-16, pages 230–239, New York, NY, USA, 2016. ACM.

[41] G. Cledou and L. S. Barbosa. Modeling families of public licensing services: A case study. In *Proceedings of the 5th International FME Workshop on Formal Methods in Software Engineering*, FormaliSE '17, pages 37–43, Piscataway, NJ, USA, 2017. IEEE Press.

[42] G. Cledou, E. Estevez, and L. S. Barbosa. A taxonomy for planning and designing smart mobility services. *Government Information Quarterly*, 35(1):61 – 76, 2018. Internet Plus Government: Advancement of Networking Technology and Evolution of the Public Sector.

[43] G. Cledou, J. Proença, and L. S. Barbosa. A refinement relation for families of timed automata. In S. Cavalheiro and J. Fiadeiro, editors, *Formal Methods: Foundations and Applications*, pages 161–178, Cham, 2017. Springer International Publishing.

[44] G. Cledou, J. Proença, and L. Soares Barbosa. Composing families of timed automata. In M. Dastani and M. Sirjani, editors, *Fundamentals of Software Engineering*, pages 51–66, Cham, 2017. Springer International Publishing.

[45] P. Clements and L. Northrop. *Software product lines*. Addison-Wesley, 2002.

[46] M. Cordy. *Model Checking for the Masses*. PhD thesis, 2014.

[47] M. Cordy, P.-Y. Schobbens, P. Heymans, and A. Legay. Behavioural modelling and verification of real-time software product lines. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*, pages 66–75. ACM, 2012.

[48] M. Cordy, P. Y. Schobbens, P. Heymans, and A. Legay. Beyond Boolean product-line model checking: Dealing with feature attributes and multi-features. *Proceedings - International Conference on Software Engineering*, pages 472–481, 2013.

[49] E. Daniel and J. Ward. Integrated service delivery: exploratory case studies of enterprise portal adoption in uk local government. *Business Process Management Journal*, 12(1):113–123, 2006.

[50] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed i/o automata: A complete specification theory for real-time systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '10, pages 91–100, New York, NY, USA, 2010. ACM.

[51] S. M. Davis. *Future perfect*. Basic Books, 1997.

[52] L. de Alfaro and T. A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, Sept. 2001.

[53] L. de Alfaro and T. A. Henzinger. *Interface-Based Design*, pages 83–104. Springer Netherlands, Dordrecht, 2005.

[54] Department of Transport Tourism and Sport. Transport Operator License - Guidelines and Forms. https://www.rtol.ie/rtol-online/forms. Accessed: 2015-05-20.

[55] A. V. Deursen. Where to Go in the Near Future : Diverging Perspectives on Online Public Service Delivery. In *Electronic Government*, pages 143–154. Springer Berlin Heidelberg, 2007.

[56] A. V. Deursen, J. V. Dijk, and W. Ebbers. Why E-government Usage Lags Behind : Explaining the Gap between Potential and Actual Usage of Electronic Public Services in the Netherlands. In *Electronic Government*, pages 269–280. Springer Berlin Heidelberg, 2006.

[57] Deutsche Gesellschaft für Internationale Zusammenarbeit. Urban Transport and Energy Efficiency. 2012.

[58] S. Djahel, N. Smith, S. Wang, and J. Murphy. Reducing Emergency Services Response Time in Smart Cities : An Advanced Adaptive and Fuzzy Approach. (October), 2015.

[59] C. Dobre and F. Xhafa. Intelligent services for Big data science. *Future Generation Computer Systems*, 37:267–281, 2014.

[60] V. Douwe, E. Estevez, A. Ojo, and T. Janowski. Software infrastructure for eŋgovernment ŋ- Eŋ-appointment service. *Communications in Computer and Information Science*, 62:141–152, 2009.

[61] E. Estevez. *Programmable Messaging for Electronic Government*. PhD thesis, Universidad Nacional del Sur, 2009.

[62] E. Estevez. EGOV Infrastructure and Services, Executive Training for Government Information Officers, Module 5, 2014.

[63] E. Estevez and T. Janowski. Building a dependable messaging infrastructure for electronic government. In *2nd International Conference on Availability, Reliability and Security, ARES*, pages 948–955, 2007.

[64] European Commission. EU eGovernment Action Plan 2016-2020. https://ec.europa.eu/digital-single-market/en/european-egovernment-action-plan-2016-2020. Accessed: 2018-05-17.

[65] European Commission. European Interoperability Framework for Pan-European eGovernment Services, 2004.

[66] EY and Danish Technological Institute. *Study on eGovernment and the Reduction of Administrative Burden*. 2014.

[67] M. Fang and C. Jianping. A Novel System for Interactive Mobile Multimedia Service in Public Transports. pages 867–870, 2013.

[68] B. Fazenda, H. Atmoko, F. G. F. Gu, L. G. L. Guan, and a. Ball. Acoustic based safety emergency vehicle detection for intelligent transport systems. *2009 Iccas-Sice*, (1):4250–4255, 2009.

[69] J. Firnkorn and M. Müller. What will be the environmental effects of new free-floating car-sharing systems? The case of car2go in Ulm. *Ecological Economics*, 70(8):1519–1528, 2011.

[70] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, P.-M. Natasa, and E. Meijers. Smart cities Ranking of European medium-sized cities. *October*, 16(October):13–18, 2007.

[71] P. Gora. A genetic algorithm approach to optimization of vehicular traffic in cities by means of configuring traffic lights. *Studies in Computational Intelligence*, 369:1–10, 2011.

[72] P. Gora and P. Wasilewski. Adaptive System for Intelligent Traffic Management in Smart Cities. In D. Ślęzak, G. Schaefer, S. Vuong, and Y.-S. Kim, editors, *Active Media Technology SE - 44*, volume 8610 of *Lecture Notes in Computer Science*, pages 525–536. Springer International Publishing, 2014.

[73] P. Gottschalk. Maturity levels for interoperability in digital government. *Government Information Quarterly*, 26(1):75–81, Jan. 2009.

[74] D. F. Gray. *Introduction to the Formal Design of Real Time Systems*. Springer-Verlag London, 1999.

[75] M. L. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the rseb. In *Proceedings. Fifth International Conference on Software Reuse (Cat. No.98TB100203)*, pages 76–85, Jun 1998.

[76] R. E. Hall, J. Braverman, J. Taylor, and H. Todosow. The Vision of A Smart City. In *2nd International Life Extension Technology Workshop*, Paris, France, 2000.

[77] J. Harper, R. Fuller, D. Sweeney, and T. Waldmann. Human factors in technology replacement: A case study in interface design for a public transport monitoring system. *Applied Ergonomics*, 1998.

[78] C. Harrison, B. Eckman, R. Hamilton, P. Hartswick, J. Kalagnanam, J. Paraszczak, and P. Williams. Foundations for Smarter Cities. *IBM Journal of Research and Development*, 54(4):1–16, jul 2010.

[79] G. Hinterwälder, C. T. Zenger, F. Baldimtsi, A. Lysyanskaya, C. Paar, and W. P. Burleson. Efficient e-cash in practice: NFC-based payments for public transportation systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7981 LNCS:40–59, 2013.

[80] G.-J. Horng, J.-P. Li, and S.-T. Cheng. Traffic congestion reduce mechanism by adaptive road routing recommendation in smart city. *2013 3rd International Conference on Consumer Electronics, Communications and Networks*, pages 714–717, nov 2013.

[81] Instituto da Mobilidade e dos Transportes IP. Carreiras de Serviço Público - Guidelines and Forms. http://www.imtt.pt/sites/IMTT/Portugues/TransportesRodoviarios/TransportePublicoPassageiros/CarreirasServicoPublico/Paginas/CarreirasdeServi{ç}oPublico.aspx. Accessed: 2015-05-20.

[82] Instituto da Mobilidade e dos Transportes IP. Licenciamento de Empresas - Guidelines and Forms. http://www.imtt.pt/sites/IMTT/Portugues/TransportesRodoviarios/TransportePublicoPassageiros/LicenciamentoEmpresas/Paginas/LicenciamentoEmpresas.aspx. Accessed: 2015-05-20.

[83] Instituto da Mobilidade e dos Transportes IP. Licenciamento de Veículos - Guidelines and Forms. Accessed: 2015-11-01.

[84] Ireland's National Police Service. Licensing of Large Public Service Vehicles - Requirements. http://www.garda.ie/Controller.aspx?Page=100. Accessed: 2015-05-20.

[85] Ireland's National Transport Authority. Licensing of Public Bus Passenger Services - Guidelines and Forms. https://www.nationaltransport.ie/taxi-and-bus-licensing/bus/. Accessed: 2015-11-01.

[86] T. Janowski. Digital government evolution: From transformation to contextualization. *Government Information Quarterly*, 32(3):221–236, 2015.

[87] T. Janowski, A. Ojo, and E. Estevez. Rapid Development of Electronic Public Services — Software Infrastructure and Software Process. In *8th Annual International Digital Government Research Conference Rapid*, pages 294–295, 2007.

[88] T. Janowski, A. Ojo, and E. Estevez. Rapid development of electronic public services: Software infrastructure and software process. In *Proceedings of the 8th annual international conference on Digital government research: bridging disciplines & domains*, pages 294–295. Digital Government Society of North America, 2007.

[89] M. Janssen and E. Estevez. Lean Government and Platform-based Governance - Doing More with Less. *Government Information Quarterly*, 30(1):S1–S8, jan 2013.

[90] Z. Ji, I. Ganchev, M. O'Droma, L. Zhao, and X. Zhang. A Cloud-Based Car Parking Middleware for IoT-based Smart Cities: Design and Implementation. *Sensors*, 14(12):22372–22393, 2014.

[91] S.-S. T. Q. Jongmans and F. Arbab. Overview of thirty semantic formalisms for reo. *Sci. Ann. Comp. Sci.*, 22:201–251, 2012.

[92] T. B. Jørgensen and B. Bozeman. Public Values - An Inventory. *Adminsitration & Society*, 39(3):354–381, 2007.

[93] M. Jung, J. Weidinger, W. Kastner, and A. Olivieri. Building Automation and Smart Cities: An Integration Approach Based on a Service-Oriented Architecture. *27th International Conference on Advanced Information Networking and Applications Workshops*, pages 1361–1367, Mar. 2013.

[94] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Software Engineering Institute, Carnegie-Mellon University, 1990.

[95] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5(1):143–168, Jan. 1998.

[96] E. I. A. Kareem and A. Jantan. An intelligent traffic light monitor system using an adaptive associative memory. *International Journal of Information Processing and Management*, 2(2):23–39, 2011.

[97] C. Kästner, S. Apel, and M. Kuhlemann. Granularity in software product lines. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 311–320, New York, NY, USA, 2008. ACM.

[98] D. Kitchin, W. R. Cook, and J. Misra. A language for task orchestration and its semantic properties. In C. Baier and H. Hermanns, editors, *CONCUR 2006 – Concurrency Theory*, pages 477–491, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[99] J. Kostiainen, C. Erkut, and F. B. Piella. Design of an audio-based mobile journey planner application. *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments - MindTrek '11*, page 107, 2011.

[100] H. Kung and D. Vlah. Efficient location tracking using sensor networks. *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.*, 3, 2003.

[101] B. H. Kwasnik. The Role of Classifiction in Knowledge Representation and Discovery. *Library Trends*, 48(1):22–47, 1999.

[102] M. Ladeira, F. Michel, and L. Senna. Public Transport Monitoring and Control: The Case of Porto Alegre, Brazil. *ICTIS 2011*, pages 275–281, 2011.

[103] K. G. Larsen, U. Nyman, and A. Wasowski. Modal i/o automata for interface and product line theories. In *European Symposium on Programming*, pages 64–79. Springer, 2007.

[104] K. Layne and J. Lee. Developing fully functional E-government: A four stage model. *Government Information Quarterly*, 18(2):122–136, 2001.

[105] F. Li and B. Li. Aggregating heterogeneous services in the smart city: The practice in china. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10135 LNCS:449–458, 2017.

[106] L. Li, F.-Y. Wang, N.-N. Zheng, and Y. Zhang. Research and developments of intelligent driving behaviour analysis. *Zidonghua Xuebao/Acta Automatica Sinica*, 2007.

[107] V. M. A. d. Lima, R. M. Marcacini, M. H. P. Lima, M. I. Cagnin, and M. A. S. Turine. A generation environment for front-end layer in e-government content

management systems. In *Proceedings of the 2014 9th Latin American Web Congress*, LA-WEB '14, pages 119–123, Washington, DC, USA, 2014. IEEE Computer Society.

[108] M. Lochau, S. Mennicke, H. Baller, and L. Ribbeck. Incremental model checking of delta-oriented software product lines. *Journal of Logical and Algebraic Methods in Programming*, (1):245–267, 2016.

[109] M. Mallus, G. Colistra, L. Atzori, M. Murroni, and V. Pilloni. Dynamic carpooling in urban areas: Design and experimentation with a multi-objective route matching algorithm. *Sustainability (Switzerland)*, 9(2), 2017.

[110] M. Mannion. Using first-order logic for product line model validation. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 176–187, London, UK, UK, 2002. Springer-Verlag.

[111] J.-V. Millo, S. Ramesh, S. N. Krishna, and G. K. Narwane. Compositional verification of evolving software product lines. *arXiv preprint arXiv:1212.4258*, 2012.

[112] E. a. Morris. Should we all just stay home? Travel, out-of-home activities, and life satisfaction. *Transportation Research Part A: Policy and Practice*, 78:519–536, 2015.

[113] G. Motta, D. Sacco, A. Belloni, and L. You. A system for green personal integrated mobility: A research in progress. *Proceedings of 2013 IEEE International Conference on Service Operations and Logistics, and Informatics, SOLI 2013*, pages 1–6, 2013.

[114] G. Motta, L. You, D. Sacco, and G. Miceli. Mobility Service Systems : guidelines for a possible paradigm and a case study. In *Proceedings of 2014 IEEE International Conference on Service Operations and Logistics, and Informatics, SOLI 2014*, number Figure 1, pages 48–53, 2014.

[115] R. Muschevici, J. Proença, and D. Clarke. Feature nets: behavioural modelling of software product lines. *Software & Systems Modeling*, pages 1–26, 2015.

[116] T. Nam and T. a. Pardo. Conceptualizing Smart City with Dimensions of Technology, People, and Institutions. In *Proceedings of the 12th Annual International Digital Government Research Conference on Digital Government Innovation in Challenging Times - dg.o '11*, page 282, New York, New York, USA, 2011. ACM Press.

[117] S. Nawaz, C. Efstratiou, and C. Mascolo. ParkSense: a smartphone based sensing system for on-street parking. *Proceedings of the 19th annual international conference on Mobile computing and networking - Mobicom '13*, 2013.

[118] P. Neirotti, A. De Marco, A. C. Cagliano, G. Mangano, and F. Scorrano. Current trends in Smart City initiatives: Some stylised facts. *Cities*, 38:25–36, jun 2014.

[119] J. D. Nelson and C. Mulley. The impact of the application of new technology on public transport service provision and the passenger experience: A focus on implementation in Australia. *Research in Transportation Economics*, 39(1):300–308, 2013.

[120] R. C. Nickerson, U. Varshney, and J. Muntermann. A method for taxonomy development and its application in information systems. *European Journal of Information Systems*, 22(3):336–359, 2012.

[121] S. Noei and A. Sargolzaei. Reducing Traffic Congestion Using Geo-fence Technology : Application for Emergency Car. pages 15–20, 2014.

[122] OECD. Recommendation of the Council on Digital Government Strategies. *Public Governance and Territorial Development Directorate*, July:12, 2014.

[123] J. Opiola and S. Wilson. Bank payments in transportation - freedom and interoperability for congestion pricing. In *15th World Congress on Intelligent Transport Systems and ITS America Annual Meeting 2008*, volume 2, pages 1303–1318, 2008.

[124] S. Panichpapiboon and W. Pattara-atikom. A Review of Information Dissemination Protocols for Vehicular Ad Hoc Networks. *IEEE Communications Surveys & Tutorials*, 14(3):784–798, 2011.

[125] M. C. Penadés, P. Martí, J. H. Canós, and A. Gómez. Product Line-based customization of e-Government documents. In N. Loutas, F. Narducci, A. Ojo, M. Palmonari, C. Paris, and G. Semeraro, editors, *PEGOV 2014: Personalization in e-Government Services, Data and Applications*, volume 1181 of *UMAP 2014 Extended Proceedings*, Aalborg, Denmark, July 2014. CEUR-WS.

[126] G. Perboli, A. De Marco, F. Perfetti, and M. Marone. A new taxonomy of smart city projects. *Transportation Research Procedia*, 3(July):470–478, 2014.

[127] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[128] B. Pokriü, S. Krþo, and M. Pokriü. Augmented Reality based Smart City Services using Secure IoT Infrastructure. In *Proceedings - 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014*, 2015.

[129] F. E. Prettenthaler and K. W. Steininger. From Ownership to Service Use Lifestyle. *Ecological Economics*, 28(3):443–453, 1999.

[130] J. Proença. *Synchronous Coordination of Distributed Components*. PhD thesis, 2011.

[131] J. Sánchez, M. Galán, and E. Rubio. Applying a traffic lights evolutionary optimization technique to a real case: "Las Ramblas" area in Santa Cruz de Tenerife. *IEEE Transactions on Evolutionary Computation*, 12(1):25–40, 2008.

[132] G. Santos, H. Behrendt, and A. Teytelboym. Part II: Policy instruments for sustainable road transport. *Research in Transportation Economics*, 28(1):46–91, 2010.

[133] F. Schnitzler, A. Artikis, M. Weidlich, I. Boutsis, T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, V. Kalogeraki, J. Marecek, A. Gal, S. Mannor, D. Kinane, and D. Gunopulos. Heterogeneous Stream Processing and Crowdsourcing for Traffic Monitoring: Highlights. In T. Calders, F. Esposito, E. Hüllermeier, and R. Meo, editors, *Machine Learning and Knowledge Discovery in Databases SE - 49*, volume 8726 of *Lecture Notes in Computer Science*, pages 520–523. Springer Berlin Heidelberg, 2014.

[134] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Comput. Netw.*, 51(2):456–479, Feb. 2007.

[135] T. Semertzidis, K. Dimitropoulos, A. Koutsia, and N. Grammalidis. Video sensor network for real-time traffic monitoring and surveillance. *IET Intelligent Transport Systems*, 4(2):103, 2010.

[136] M. Seredynski, P. Ruiz, K. Szczypiorski, and D. Khadraoui. Improving Bus Ride Comfort Using GLOSA-based Dynamic Speed Optimisation. In *Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS*, 2014.

[137] C. Sommer, O. K. Tonguz, and F. Dressler. Traffic information systems: Efficient message dissemination via adaptive beaconing. *IEEE Communications Magazine*, 49(5):173–179, 2011.

[138] D. H. Stolfi and E. Alba. Red Swarm: Reducing travel times in smart cities by using bio-inspired algorithms. *Applied Soft Computing Journal*, 24:181–195, 2014.

[139] T. Suzumura, S. Kato, T. Imamichi, M. Takeuchi, H. Kanezashi, T. Ide, and T. Onodera. X10-based massive parallel large-scale traffic flow simulation. *Proceedings of the ACM SIGPLAN 2012 X10 Workshop on - X10 '12*, pages 1–4, 2012.

[140] R. Szabo, K. Farkas, M. Ispany, A. A. Benczur, N. Batfai, P. Jeszenszky, S. Laki, A. Vagner, L. Kollar, C. Sidlo, R. Besenczi, M. Smajda, G. Kover, T. Szincsak, T. Kadek, M. Kosa, A. Adamko, I. Lendak, B. Wiandt, T. Tomas, A. Z. Nagy, and G. Feher. Framework for smart city applications based on participatory sensing. In *Cognitive Infocommunications (CogInfoCom), 2013 IEEE 4th International Conference on*, pages 295–300, dec 2013.

[141] S. Tarapiah, S. Atalla, L. Autonomo, and B. Alsayid. Smart On-Board Transportation Management System Geo-Casting Featured. In *2014 World Congress on Computer Applications and Information Systems, WCCAIS 2014*, 2014.

[142] C. Tautz and C. Gresse von Wangenheim. REFSENO : A Representation Formalism for Software Engineering Ontologies. Technical Report 015, 1998.

[143] D. Teodorović and P. Lučić. Intelligent parking systems. *European Journal of Operational Research*, 175(3):1666–1681, 2006.

[144] T. Thum, D. Batory, and C. Kastner. Reasoning about edits to feature models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 254–264, Washington, DC, USA, 2009. IEEE Computer Society.

[145] UNEP. *Sustainable, Resource Efficient Cities Making it Happen!* 2012.

[146] United Nations. *World Urbanization Prospects: The 2014 Revision, Highlights.* 2014.

[147] U.S. Environmental Protection Agency. Inventory of U.S. Greenhouse Gas Emissions and Sinks: 1990 - 2013. Technical report, 2015.

[148] W. M. van Der Aalst, A. H. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.

[149] D. Washburn, U. Sindhu, S. Balaouras, R. A. Dines, N. M. Hayes, and L. E. Nelson. Helping CIOs Understand Smart City Initiatives, 2010.

[150] M. a. Wimmer. A European perspective towards online one-stop government: the eGOV project. *Electronic Commerce Research and Applications*, 1(1):92–103, Mar. 2002.

[151] L. Wischhof, A. Ebner, and H. Rohling. Information dissemination in self-organizing intervehicle networks. *IEEE Transactions on Intelligent Transportation Systems*, 6(1):90–101, 2005.

[152] T. Yahiaoui, L. Khoudour, and C. Meurie. Real-time passenger counting in buses using dense stereovision. *Journal of Electronic Imaging*, 19(3):031202, 2010.

[153] X. Yang and L. Zhang. A Dynamic Method to Monitor Public Transport based on. In *17th International Conference on Intelligent Transportation Systems (ITSC)*, pages 924–929, 2014.

[154] I. Yaqoob, I. A. T. Hashem, Y. Mehmood, A. Gani, S. Mokhtar, and S. Guizani. Enabling communication technologies for smart cities. *IEEE Communications Magazine*, 55(1):112–120, 2017.

[155] S. E. Yoo, P. K. Chong, T. Park, Y. Kim, D. Kim, C. Shin, K. Sung, and H. Kim. DGS: Driving guidance system based on wireless sensor network. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, pages 628–633, 2008.

[156] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: Driving directions based on taxi trajectories. *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '10*, (2010):99, 2010.