
Towards Prescriptive Analytics in Cyber-Physical Systems

Laurynas Šikšnys

Ph.D. Thesis
February 2014

Supervisors: **Prof. Torben Bach Pedersen**
Department of Computer Science
Aalborg University
DK-9220 Aalborg Ø, Denmark

Prof. Dr.-Ing. Wolfgang Lehner
Department of Computer Science
Technische Universität Dresden
01062 Dresden, Germany

A thesis submitted to the Faculty of Engineering and Science at Aalborg University and the Faculty of Computer Science at Technische Universität Dresden, in partial fulfilment of the requirements within the scope of the IT4BI-DC programme for the joint Ph.D. degree in computer science. The thesis is not submitted to any other organization at the same time.

Abstract

More and more of our physical world today is being monitored and controlled by so-called cyber-physical systems (CPSs). These are compositions of networked autonomous cyber and physical agents such as sensors, actuators, computational elements, and humans in the loop. Today, CPSs are still relatively small-scale and very limited compared to CPSs to be witnessed in the future. Future CPSs are expected to be far more complex, large-scale, wide-spread, and mission-critical, and found in a variety of domains such as transportation, medicine, manufacturing, and energy, where they will bring many advantages such as the increased efficiency, sustainability, reliability, and security. To unleash their full potential, CPSs need to be equipped with, among other features, the support for automated *planning* and *control*, where computing agents collaboratively and continuously *plan* and *control* their actions in an intelligent and well-coordinated manner to secure and optimize a physical process, e.g., electricity flow in the power grid.

In today's CPSs, the *control* is typically automated, but the *planning* is solely performed by humans. Unfortunately, it is intractable and infeasible for humans to plan every action in a future CPS due to the complexity, scale, and volatility of a physical process. Due to these properties, the control and planning has to be continuous and automated in future CPSs. Humans may only analyse and tweak the system's operation using the set of tools supporting *prescriptive analytics* that allows them (1) to make predictions, (2) to get the suggestions of the most prominent set of actions (decisions) to be taken, and (3) to analyse the implications as if such actions were taken.

This thesis considers the *planning* and *control* in the context of a large-scale multi-agent CPS. Based on the smart-grid use-case, it presents a so-called PRESCRIPTIVECPS – which is (the conceptual model of) a multi-agent, multi-role, and multi-level CPS automatically and continuously taking and realizing decisions in near real-time and providing (human) users *prescriptive analytics* tools to analyse and manage the performance of the underlying physical system (or process). Acknowledging the complexity of CPSs, this thesis provides contributions at the following three levels of scale: (1) the level of a (full) PRESCRIPTIVECPS, (2) the level of a single PRESCRIPTIVECPS agent, and (3) the level of a component of a CPS agent software system.

At the *CPS level*, the contributions include the definition of PRESCRIPTIVECPS,

according to which it is the system of interacting *physical* and *cyber* (sub-)systems. Here, the cyber system consists of hierarchically organized inter-connected agents, collectively managing instances of so-called *flexibility*, *decision*, and *prescription* models, which are short-lived, focus on the future, and represent a *capability*, an (user's) *intention*, and *actions* to change the behaviour (state) of a physical system, respectively.

At the *agent level*, the contributions include the three-layer architecture of an agent software system, integrating the number of components specially designed or enhanced to support the functionality of PRESCRIPTIVECPS.

At the *component level*, the most of the thesis contribution is provided. The contributions include the *description*, *design*, and *experimental evaluation* of (1) a unified multi-dimensional schema for storing flexibility and prescription models (and related data), (2) techniques to incrementally aggregate flexibility model instances and disaggregate prescription model instances, (3) a database management system (DBMS) with built-in *optimization problem solving* capability allowing to formulate optimization problems using SQL-like queries and to solve them “inside a database”, (4) a real-time data management architecture for processing instances of flexibility and prescription models under (soft or hard) timing constraints, and (5) a graphical user interface (GUI) to visually analyse the flexibility and prescription model instances. Additionally, the thesis *discusses* and *exemplifies* (but provides no evaluations of) (1) domain-specific and in-DBMS generic *forecasting* techniques allowing to forecast instances of flexibility models based on historical data, and (2) powerful ways to analyse *past*, *current*, and *future* based on so-called hypothetical *what-if scenarios* and flexibility and prescription model instances stored in a database. Most of the contributions at this level are based on the smart-grid use-case.

In summary, the thesis provides (1) the model of a CPS with planning capabilities, (2) the design and experimental evaluation of *prescriptive analytics* techniques allowing to effectively *forecast*, *aggregate*, *disaggregate*, *visualize*, and *analyse* complex models of the physical world, and (3) the use-case from the energy domain, showing how the introduced concepts are applicable in the real world. We believe that all this contribution makes a significant step towards developing planning-capable CPSs in the future.

Thesis Details

Thesis Title: Towards Prescriptive Analytics in Cyber-Physical Systems
Ph.D. Student: Laurynas Šikšnys
Supervisors: Prof. Torben Bach Pedersen, Aalborg University
Prof. Dr.-Ing. Wolfgang Lehner, Technische Universität Dresden

The main body of this thesis is based on the following papers:

- [1] M. Boehm, L. Dannecker, A. Doms, E. Dovgan, B. Filipič, U. Fischer, W. Lehner, T. B. Pedersen, Y. Pitarch, L. Šikšnys, T. Tušar: “Data Management in the MIRABEL Smart Grid System”, in *Proc. of EDBT/ICDT Workshops*, 2012 (Input for Chapter 2).
- [2] L. Šikšnys, C. Thomsen, T. B. Pedersen: “MIRABEL DW: Managing Complex Energy Data in a Smart Grid”, in *Proc. of DaWaK*, 2012 (Input for Chapter 5).
- [3] L. Šikšnys, M. E. Khalefa, T. B. Pedersen: “Aggregating and Disaggregating Flexibility Objects”, in *Proc. of SSDBM*, 2012 (Input for Chapter 6).
- [4] T. Tušar, L. Šikšnys, T. B. Pedersen, E. Dovgan, B. Filipič: “Using aggregation to improve the scheduling of flexible energy offers”, in *Proc. of BIOMA*, 2012 (Input for Chapter 6).
- [5] U. Fischer, L. Dannecker, L. Šikšnys, F. Rosenthal, M. Boehm, W. Lehner: “Towards Integrated Data Analytics: Time Series Forecasting in DBMS”, *Datenbank-Spektrum*, vol. 13, no. 1, 2013 (Input for Chapter 7).
- [6] D. Kaulakienė, L. Šikšnys, Y. Pitarch: “Towards the Automated Extraction of Flexibilities from Electricity Time Series”, in *Proc. of EDBT/ICDT Workshops*, 2013 (Input for Chapter 7).
- [7] L. Šikšnys, T. B. Pedersen: “SolveDB: Solving Optimization Problems In the Database”, (in submission) (Input for Chapter 8).

- [8] U. Fischer, D. Kaulakienė, M. E. Khalefa, W. Lehner, T. B. Pedersen, L. Šikšnys, C. Thomsen: “Real-time Business Intelligence in the MIRABEL Smart Grid System”, in *Proc. of BIRTE*, 2012 (Input for Chapter 10).
- [9] L. Šikšnys, D. Kaulakienė: “Visualizing Complex Energy Planning Objects With Inherent Flexibilities”, in *Proc. of EDBT/ICDT Workshops*, 2013 (Input for Chapter 11).

This thesis has been submitted for assessment in partial fulfilment of the joint Ph.D. degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty of Engineering and Science at Aalborg University and the Faculty of Computer Science at Technische Universität Dresden. The thesis is not in its present form acceptable for open publication but only in limited and closed circulation as copyright may not be ensured.

Acknowledgements

This thesis would not have been possible without the help and support of many people, whom I wish to acknowledge in this section.

First, I would like to thank my academic supervisor professor Torben Bach Pedersen for giving me this research opportunity and for guiding and supporting me throughout the entire Ph.D. study. I admire his creativity, mental discipline, and efficiency at work. I am very grateful for his time, dedication, and patience needed for teaching me how to write scientific papers and conduct research in general. Many thanks for his elaborate comments and recommendations on the content of this thesis.

During my Ph.D. study, I visited Database Technology Group at Technische Universität Dresden for six months. I would like to thank Prof. Dr.-Ing. Wolfgang Lehner for providing creative and stimulating environment at his research group and for productive research collaboration – including his help on structuring and improving this document. Additionally, thanks to my office mates in Dresden - Ulrike Fischer and Benjamin Schlegel - for being such inspiring, supportive, and entertaining during my complete study abroad.

I would like to thank all my colleagues and administrative staff at the Database and Programming Technologies group at Aalborg University for friendly and helpful atmosphere and a productive collaboration. Special thanks to Kasper Søren Luckow and Katja Hose for helping me with the translations of this thesis summary. Also, thanks for the productive and enjoyable collaboration to my international partners from SAP Research Dresden, Dresden University of Technology, and Jožef Stefan Institute.

On a more personal note, my studies would not have been possible without the support and continuous encouragement from my family and friends.

Finally, I would like to acknowledge the financial support of the Faculty of Engineering and Science, Aalborg University and the MIRABEL project, which was funded by the European Commission under grant agreement number 248195.

Contents

Abstract	iii
Thesis Details	v
Acknowledgements	vii
1 Introduction	1
1.1 Cyber-Physical Systems	1
1.2 Planning and Control	3
1.3 Prescriptive Analytics and Automation	4
1.4 Multi-Agent Planning	6
1.5 Problem Statement	9
1.6 Thesis Contributions	9
1.7 Thesis Overview	14
2 The MIRABEL Use-case	17
2.1 MIRABEL Project Overview	17
2.2 MIRABEL ICT System	20
2.3 Example of Electric Vehicle Charging	22
2.4 Summary	23
3 Cyber-Physical System for Planning and Control	25
3.1 Definition of PrescriptiveCPS	25
3.2 Models of a Physical System	27
3.3 Core Planning Operations with Flexibility, Decision, and Prescription Models	32
3.4 Planning Workflow of PrescriptiveCPS	35
3.5 Agent Roles in PrescriptiveCPS	37
3.6 MIRABEL ICT System as the Instance of PrescriptiveCPS	39
3.7 Common Requirements of PrescriptiveCPS	43

3.8	Summary	47
4	Agent Software System for Prescriptive Analytics	49
4.1	Architecture of an Agent Software System	49
4.2	Analytical Capabilities of the DBMS	51
4.3	Queries Supported by the DBMS	52
4.4	Contribution on the Agent Components	55
4.5	Summary	56
5	Multi-Dimensional Flexibility and Prescription Model Representation	57
5.1	Data Storage in the MIRABEL Use-case	58
5.2	Modelling of Different Actors and Roles	59
5.3	Modelling of Flex-offers and Flex-offer assignments	61
5.4	Modelling of Time Series	65
5.5	Full Data Warehouse Schema	67
5.6	Performance Study	69
5.6.1	The MIRABEL DW schema alternatives	69
5.6.2	The experimental setup	70
5.6.3	Experiments with flex-offers (Q1)	70
5.6.4	Experiments with flex-offer assignments (Q2 and Q3)	71
5.6.5	Experiments with time series (Q4 and Q5)	73
5.6.6	Summary	74
5.7	Related Work	75
5.8	Summary and Discussion	76
6	Flexibility Model Aggregation	77
6.1	Flexibility Model Aggregation in MIRABEL	77
6.2	Problem of Flex-offer Aggregation	78
6.3	N-to-1 Aggregation Technique	83
6.4	N-to-M Aggregation Technique	86
6.5	Incremental Aggregation Technique	88
6.6	Performance Study	91
6.6.1	Stand-alone aggregation study	91
6.6.2	Integrated aggregation study	95
6.7	Related Work	99
6.8	Summary and Discussion	100
7	Flexibility Model Forecasting	101
7.1	Invariant Flexibility Model Forecasting in DBMS	101
7.1.1	Built-in forecasting in MIRABEL	102
7.1.2	Requirements and related work	103
7.1.3	The architecture of a DBMS for forecasting	104

7.1.4	Processing forecast queries	110
7.1.5	Research topics & challenges	113
7.1.6	Summary	114
7.2	Variant Flexibility Model Forecasting	115
7.2.1	Generic flex-offer generation architecture	115
7.2.2	Flex-offer generation approaches	116
7.2.3	Summary	118
7.3	Summary and Discussion	119
8	Decision Model Solving in a DBMS	121
8.1	Built-in Optimization Problem Solving	122
8.2	Solve Queries	126
8.3	SolveDB Translation Workflow	128
8.3.1	The three-level adaptation workflow	128
8.3.2	The multi-level abstraction workflow	129
8.3.3	Relational solve operator	131
8.4	Solve Query Formulation	131
8.4.1	Extending SQL for optimization problems	131
8.4.2	User-friendly view descriptors	132
8.5	Solve Query Processing	133
8.5.1	Overall solve query processing workflow	133
8.5.2	The workflow of an atomic view solver	134
8.5.3	The workflow of a composite view solver	137
8.5.4	Summary of solve query processing	138
8.6	Solve Query Optimization	139
8.7	Architecture of SolveDB	140
8.8	Experimental Evaluation	141
8.8.1	PostgreSQL-based implementation	142
8.8.2	Results of the experimental evaluation	143
8.9	Related Work	148
8.10	Summary and Discussion	149
9	Flexibility and Prescription Model What-If Analysis in a DBMS	151
9.1	What-If Analysis and Related Work	151
9.2	SQL Extension for What-If Scenarios	152
9.3	What-If Queries in the MIRABEL Use-Case	154
9.3.1	Hypothetical scenarios in measurement analysis	154
9.3.2	Hypothetical scenarios in energy planning	155
9.3.3	Time travelling in energy planning	157
9.4	Summary and Discussion	158

10 Flexibility, Decision, and Prescription Model Processing in Real-Time	159
10.1 Analytical and Operational Agent Workflows	159
10.2 Planning Component Architecture for Real-time Processing	160
10.3 Data Storage Optimizations	163
10.4 Approximate Query Processing	164
10.5 Subscriptions and Incremental Planning	166
10.6 Experimental Results	168
10.6.1 Experiments on dedicated storage	169
10.6.2 Experiments on aggregation input caching	169
10.6.3 Experiments on forecast model maintenance	171
10.7 Summary and Discussion	172
11 Flexibility and Prescription Model Visualization	173
11.1 Model Visualization and Analysis in MIRABEL	173
11.2 Initial Results When Implementing Visual Model Analysis Framework .	178
11.3 Related Work	180
11.4 Summary and Discussion	182
12 Conclusion and Future Research Directions	185
12.1 Contributions	186
12.2 Discussion	188
12.3 Future Work	191
Dansk Resumé (Summary in Danish)	193
Zusammenfassung (Summary in German)	195
Bibliography	196
Statement of Authorship	209

Chapter 1

Introduction

1.1 Cyber-Physical Systems

Today, more and more of our physical world is being monitored and controlled by so-called *cyber-physical systems* (CPSs) featuring highly integrated computational and physical capabilities. CPSs are compositions of networked autonomous *cyber-agents* (e.g., sensors, actuators, computational elements) and humans that monitor and influence the operation of the system (see Figure 1.1). Such systems can be found in areas as diverse as aerospace, automotive, healthcare, manufacturing, transportation, entertainment, and energy where they bring many advantages such as increased efficiency, sustainability, reliability, and security.

The CPSs today are only at the pre-matured state, still relatively small-scale, and lacking many features and the potential of the full-fledged CPSs to be witnessed in the future. The CPSs of tomorrow are expected to be far more complex, large-scale, mission-critical, and wide-spread compared to the existing CPSs today. The CPSs of tomorrow will be systems of systems encompassing large areas such as country, continent, or the whole Earth, coordinating physical processes both at the nano and the mega scale, and dealing with thousands or millions of agents such as sensors, actuators, computational elements, and humans in the loop, where they either control or take part in the system. Computational and communication capabilities of CPSs will be embedded in nearly all types of objects and structures such as a human body, households, vehicles, or road infrastructure, and new types applications harnessing these capabilities will offer enormous economic benefit and social impact in variety of domains. For the vision of future CPSs to become reality, many challenges with the regards to adaptability, autonomy, efficiency, functionality, reliability, safety, and usability needs to be addressed today, requiring inputs from engineering, mathematics, computer science and other fields, combined with domain-specific knowledge.

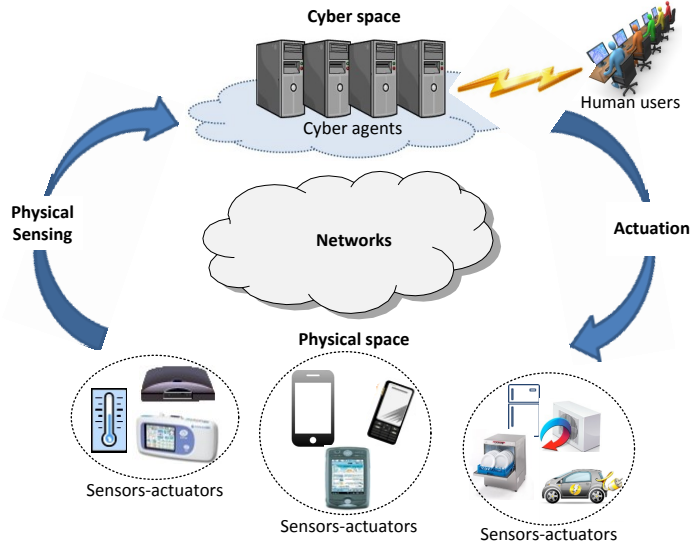


Figure 1.1: The components of cyber-physical systems

One of the major challenges in the context of future CPSs is the support for the intelligent and automated *planning and control*, as part of which computational agents of a CPS collaboratively *plan* and *control* their actions in a well-coordinated intelligent manner, potentially, at different time scales (long-, short-term) and aggregation levels (nano, micro, and macro) to secure and optimize a managed physical process, e.g., electricity flow in the grid. In today's CPSs, the control is typically automated but the planning is solely performed by humans, often with the aid from *decision support tools* (e.g., OLTP, OLAP, data mining, and optimization software systems). In a future CPS, it becomes intractable and infeasible for humans to plan actions of a CPS due to its complexity and the scale as well as the volatility of the physical process. Consequently, new generation CPSs need a support for automated planning and control, where computational agents automatically and continuously perform planning and control activities, and humans only tweaks the system's operation at coarser time granularity and higher data aggregation level, by utilizing a support from a CPS. When the planning and control are done intelligently and effectively, CPSs can indeed achieve the ultimate CPS goals [10], among which are zero-energy footprint buildings and cities, extreme-yield agriculture, accident-free transportation systems, and blackout-free electricity.

1.2 Planning and Control

In the general sense, *planning* and *control* are the two essential activities of an intelligent behaviour during which decisions are made and acted upon. To define and show the relationship between planning and control, it is convenient to use a simple conceptual model [11] (see Figure 1.2a) defining interaction between three components:

1. A *dynamic system* that evolves according to the *events* and *actions* that it receives.
2. A *controller* that produces *actions* according to observations and a *plan*, which is a structure that describes appropriate actions.
3. A *planner* that synthesizes a plan for the controller in order to achieve *objectives* based on *context information* such as a description of the dynamic system, the initial situation, external data, etc.

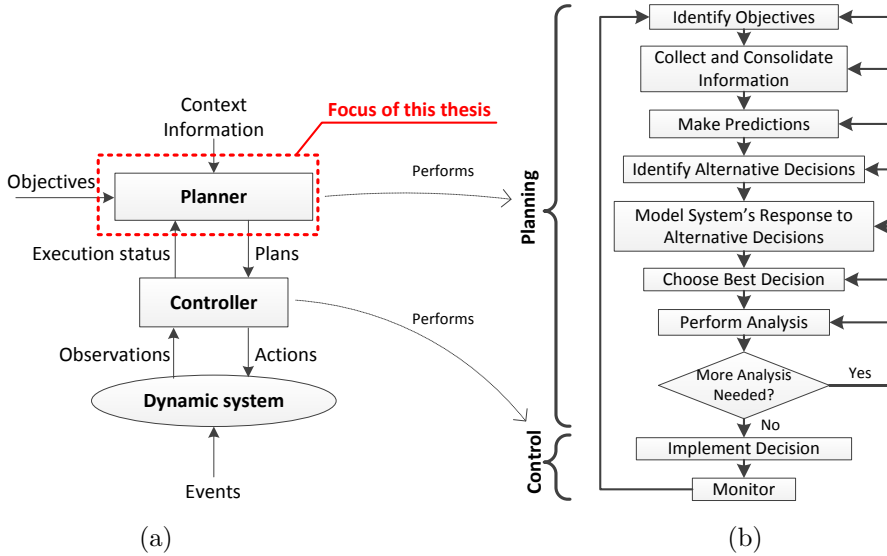


Figure 1.2: The conceptual model (a) and the process of planning and control (b)

The planner and the controller are entities such as humans or *agents* – autonomous interacting hardware- and software-based computer systems [12]. With the respect to the conceptual model, *planning* and *control* lead to (effective) *plans* and *actions* and are two closely related processes (activities) performed by the planner and the controller, respectively. Here, *planning* is a rational, systematic, and (often) cognitive process looking ahead in time (in short- or long-term) to determine the future course of action in

order to achieve the objectives laid down. On the contrary, *control* looks at current time (or backwards in time) to find out if the dynamic system is behaving (or behaved) according to the plan (course), and brings the system back to the desired state in the case of (plan) deviations. The generalized joint process of planning and control is depicted in Figure 1.2b. Among others, it includes the steps of (1) information collection and consolidation, (2) prediction of the future behaviour, (3) alternative decision identification, (4) the selection of the best decision and (5) the implementation of the decision.

1.3 Prescriptive Analytics and Automation

In most planning and control scenarios today, control is typically automated, and planning is solely performed by humans, mostly because human-planners, even though they are slower than machines, have better cognitive capabilities required for planning.

To aid humans in planning, the vast amount of *Decision Support Systems* (DSSs) – ranging from simple data querying to scalable machine learning systems – was developed over the last five decades. Most of these systems focus on one or multiple steps of planning (see Figure 1.3) and are designed exclusively for business enterprises, where they support the planning of business process through the exploration, prediction, and/or actuation of the business performance. Depending on their capabilities, these systems support one of the following types of analytics, determining their usefulness (value), difficulty, and the level of intelligence (see Figure 1.4):

Descriptive analytics focuses on the data collection and consolidation step (see Figure 1.3). It is the most common and well-understood type of business analytics [13]. It categorizes, consolidates, and summarizes data to convert it into useful information for the purposes of understanding and analysing past and current business performance. The typical question that the descriptive analytics helps answering is: “What has happened in the past or is happening now and why?”.

Predictive analytics focuses on the data collection/consolidation and the predictions steps (see Figure 1.3). It predicts future by examining historical data, detecting patterns or relationships in data, and extrapolating these relationships forward in time. By using advanced techniques and prediction models, the predictive analytics can help to detect trends and hidden patterns in large quantities of data. “What will happen in the future?” – is a typical question that the predictive analytics helps to answer.

Prescriptive analytics focuses on nearly all steps of planning (see Figure 1.3), including the steps of data collection/consolidation, prediction, and the decision making. It is a new and emerging type of business analytics [13], aiming to recommend and present the best course of decisions (actions) to take advantage of the predictions in a timely manner. It allows what-if analysis [14] and generates effective decisions in the form of *prescriptions* – short-term plans of what to do next in the present situation. To derive such prescriptions, as well as to explore factors, constraints, and the interactions between them, prescriptive analytics encourages [15] the use of *decision models* and

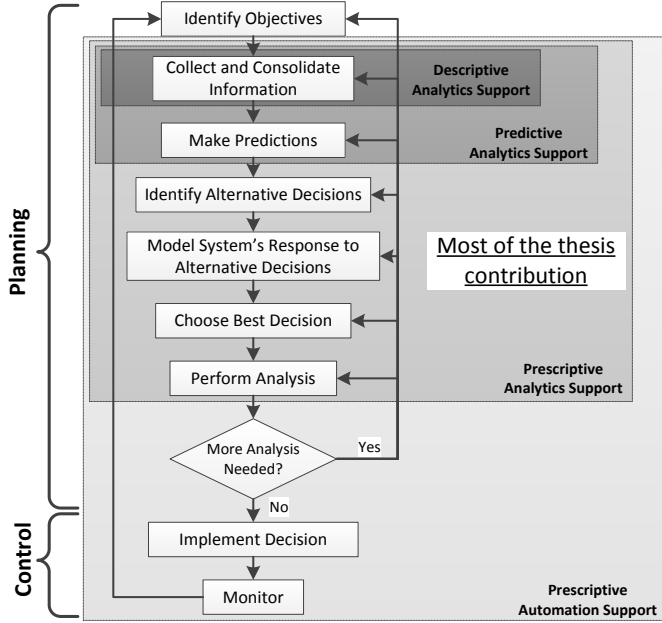


Figure 1.3: The process of planning and control and the level of DSS support

mathematical optimization, allowing to search for the best set of prescriptions among feasible candidates. Prescriptive analytics addresses the question: “What should we do to achieve our goal?”.

Unlike in traditional scenarios where humans conduct planning (with the support from DSS), the future CPS scenario requires fully automated planning and control where humans only monitor the planning and plan realization process, and, when needed, are able to re-program or assist machines, e.g., by utilizing the provided DSS tools. In other words, CPSs have to become so-called *Decision Automation Systems* (DASs) eliminating the need for humans in planning and control (e.g., Google Car [16]). With the support of *prescriptive automation*, DASs are able to automate the complete planning and control process (shown in Figure 1.3) by offering an elaborate and built-in control (software, hardware, firmware, and more) to automatically “action” prescriptions after they are generated automatically by *prescriptive analytics* tools or manually by (human) users.

However, fully automated control and planning is challenging. Typically, the automated control is simpler than the automated planning, and, consequently, various automated control approaches (e.g., *open-loop* and *closed-loop* controllers) has been proposed

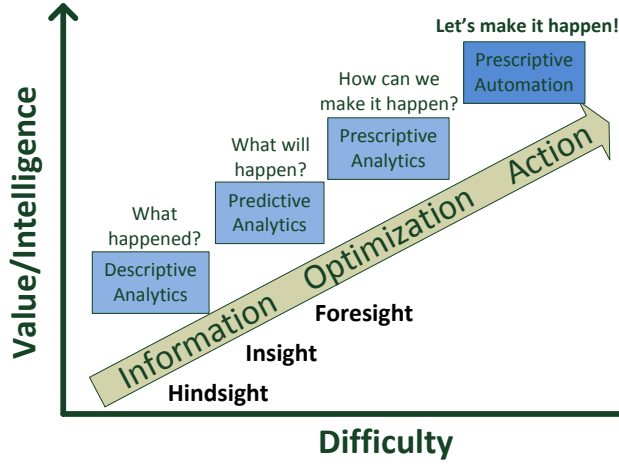


Figure 1.4: The phases of business analytics

and well studied in the field of *control theory* [17]. *Automated planning*, on the other hand, is difficult even in simple deterministic single-agent scenarios, which are the focus of the research fields of *classical and neoclassical planning* [11]. In the CPS setting, automating the planning becomes even more challenging as the planning activities have to be done by a large number of agents managing the physical world that is complex, volatile, and non-deterministic. We now show how planning in the CPS setting differs compared to the planning in other multi-agent scenarios, previously explored in the field of *multi-agent planning*.

1.4 Multi-Agent Planning

In the multi-agent setting, agents have to share resources, coordinate activities, and aim to achieve goals that are common to the whole agent population and/or are specific to individual agents. The field of *multi-agent planning* focuses on planning in such multi-agent scenarios and distinguishes the following three types of distributed planning [18]:

Cooperative Distributed Planning (CSP) distributes the *single* planning process (shown in Figure 1.2) among two or more agents cooperating to build a common global plan using some parallel planning algorithm. Agents participate in the construction of the plan and exchange information about their individual sub-

plans, which are first constructed and later adjusted to the needs of a global common plan realization.

Negotiated Distributed Planning (NDP) relies on *multiple* planning processes that are pursued by each agent concurrently. It aims for the successful realization of local agent goals as well as the goal of an agent group. The plan embracing a group of agents results from a negotiation and action coordination between agents.

Distributed Continual Planning (DCP) relies on *multiple* planning and control processes coupled together and seen as ongoing, dynamic, and interleaved joint activities pursued by each agent concurrently (as shown in Figure 1.2). As soon as deviations from a plan are detected, continual planning not only reacts to changes that threaten the execution of the plan, but also looks for opportunities to improve the plan.

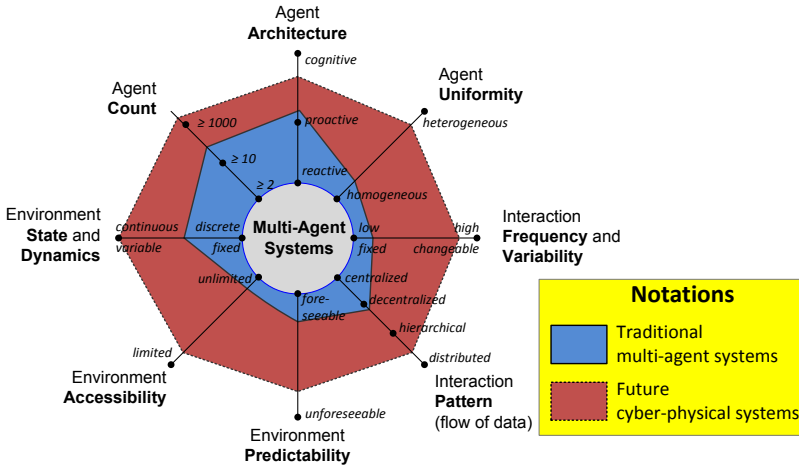


Figure 1.5: The dimensions of the multi-agent planning problem

For each of the distributed planning types, various approaches of multi-agent planning were proposed [18]. All these approaches, with no exception, make certain assumptions [19, 20] concerning the properties of *agents*, *agent interaction*, and *physical environment* (see *Traditional multi-agent systems* in Figure 1.5). Unfortunately, often these assumptions are too restrictive for the existing approaches to be practically used in the real-world [18] (including the CPS setting). Among all simplifying assumptions these approach make, there is one or more assumptions from the following list:

- **Agents** are homogeneous, reactive, and with no or limited cognitive capabilities. They pursue distributed planning (*CSP*, *NDP*, or *DCP*) to achieve the goals that

are either contradicting or complementary. The number of agents is (relatively) small.

- **Interaction** (communication) between agents is fixed, reliable, having centralized (or decentralized) pattern, and based on the signal passing (rather than the knowledge passing).
- **Environment** in which agents act is discrete (has a countable number of states) and can be accurately observed (accessible) and predicted (predictable) by agents. It changes only due to agents' action having one predictable outcome at any given moment (deterministic).

These simplifying assumptions are much too restrictive for the CPS setting, characterized as following:

- **Agents** are heterogeneous, reactive, proactive, and with cognitive capabilities where humans and machines collaborate to formulate more efficient and robust plans (aka. *mixed-initiative planning* [21,22]). There are thousands of such agents, and they pursue *distributed continual planning* (DCP) in real- or near real-time to achieve their goals that are both contradicting and complementary within distinct groups of agents. As the part of the planning, complex models (e.g., mathematical, forecasting) need to be built, decision (optimization) problems need to be solved, and decision needs to be followed [23,24].
- **Interaction** (communication) between agents is frequent (due to DCP), unreliable, knowledge-intensive, and changeable in configuration (due to the agent *entering* and *leaving* a CPS participation). The interaction between agents follows a distributed hierarchical pattern reflecting the organizational view of the physical world (e.g., consider enterprise's hierarchical organization that includes a *president*, *vice president*, *manager*, and *employee*).
- **Environment** is a physical system or process in the real-world. It is dynamic, continuous, and evolving in ways beyond the agent's control. Its state cannot be accurately observed (limited accessibility) and predicted (predictions are possible, but only at higher aggregation levels). Furthermore, the actions of an agent may have several predictable outcomes (non-deterministic), and they are often mission critical and with strict timing requirements that must not be violated.

Consequently, the accomplishments of the multi-agent research, though fine within their intended settings, are not sufficient for the setting of future CPSs. New multi-agent planning approaches are required to support the automated planning in the CPS setting. Motivated by this challenge, this thesis focuses on the future CPS setting and aims to design (1) a CPS, at multiple levels of scale, for mixed-initiative distributed continuous planning (DCP) as well as (2) prescriptive analytics tools and technology for such a CPS. The next section elaborates the overall problem addressed in this thesis.

1.5 Problem Statement

In summary, future CPSs will be much more integrated into the physical world, where they will manage various physical systems and processes in a secure and optimal manner and thus bring many advantages to the society. For that to become a reality, CPSs have to become *decision support and automation systems* (DSSs and DASs) with a large number of intelligent agents collectively realizing mixed-initiative *distributed continual planning* (DCP) and offering *prescriptive analytics* tools for (human) users. The users must be able to use these tools to monitor the plan realization and to tweak the processes of planning through more effective decision models. Unfortunately, most of the ideas and approaches in the fields of *prescriptive analytics*, *prescriptive automation*, and *multi-agent planning* are either too limited for the CPS setting (see Section 1.4) or presented only at the conceptual level [15] with no common and standardized theories and techniques available. Consequently, the emerging field of CPSs requires new *prescriptive analytics*, *prescriptive automation*, and *distributed continual planning* theories and methods that can practically be applicable in the CPS setting. This leads to the following overall problem statement of this thesis:

How to (1) design a CPS at multiple levels of scale for mixed-initiative (human- and machine-based) distributed continuous planning (DCP), how to (2) apply such design in a concrete real-world use-case, and how to (3) develop a software that supports the planning and control activities (show in Figure 1.2) and offers (human) users prescriptive analytics tools for monitoring the plan realization and for tweaking the processes of planning through more effective decision models?

1.6 Thesis Contributions

This thesis looks into the problem of designing, and practically applying the design of, a multi-agent CPS for mixed-initiative distributed continual planning (DCP). To tackle this problem, the thesis presents PRESCRIPTIVECPS, which is our proposed (conceptual model of a) multi-agent, multi-role CPS where computational agents are organized hierarchically to optimally manage an underlying physical system (see Figure 1.6). The CPS takes and realizes decisions automatically, continuously, and in near real-time (DCP). Agents at different levels of the hierarchy deal with so-called *flexibility* (world), *decision*, and *prescription* models of different aggregations levels. Depending on supported model management functionality, an agent takes one or multiple *agent roles* such as the *sensor*, *aggregator*, and/or *global prescriptor* (decision-maker) role. Human users owning and/or using agents in decision-making roles are provided *prescriptive analytics* tools for monitoring plan realization (control) and for tweaking the process of planning.

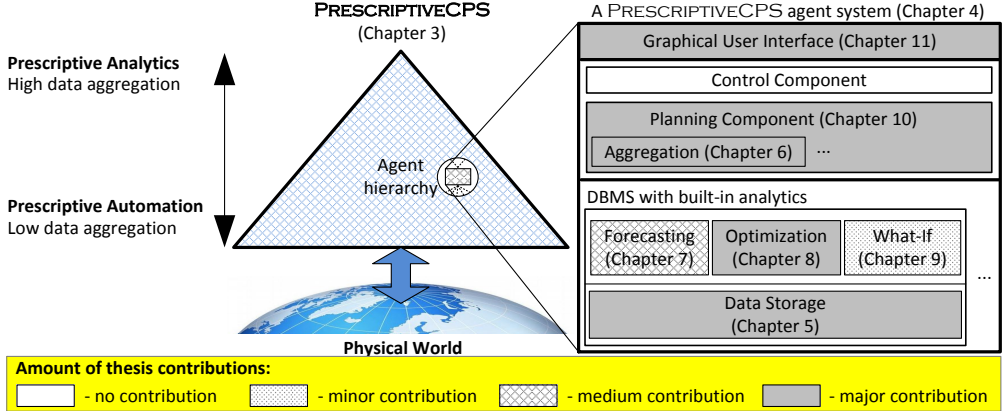


Figure 1.6: The architecture of PRESCRIPTIVECPS and thesis contributions on parts of the system

In the thesis, relevant concepts are exemplified and studied using a real-world use-case from the smart-grid domain. In the use-case, an instance of PRESCRIPTIVECPS is specialized for balancing electricity consumption and production in the power grid with a large number of renewable energy sources (e.g., windmills), as specified by the EU FP7 project MIRABEL [25] (abbr. Micro-Request-Based Aggregation, Forecasting and Scheduling of Energy Demand, Supply and Distribution).

As seen in Figure 1.6, PRESCRIPTIVECPS is the system of hierarchically organized agents, where each agent is a complex individual multi-component system. Acknowledging this complexity, the thesis focuses to the multiple levels of a PRESCRIPTIVECPS scale: (1) the level of the (full) CPS, (2) the level of a CPS agent, and (3) the level of a component of a CPS agent software system. The thesis provides contributions at all these levels. The amounts of contribution at each of these levels, together with relevant thesis chapters, are shown in Figure 1.6. The meanings of the contribution amounts used in the figure are as follows:

- No contribution** A particular entity (e.g., system or component) is only *abstractly defined* (as it might be specific to an application). No detailed discussion, examples, architecture, and experimental evaluation are given.
- Minor contribution** A particular entity is only *discussed* and *exemplified*. No architecture and the experimental evaluation of a prototype are provided.
- Medium contribution** A particular entity is *discussed* and *exemplified*, and *architecture* is presented. However, no experimental evaluation of a prototype is provided.

Major contribution A particular entity is *discussed* and *exemplified*, *architecture* is presented, and the *experimental evaluation* of the prototype is provided.

By focusing at the different levels of the CPS scale, we now elaborate these contributions. The amount of a contribution is depicted where relevant.


Thesis contributions at the CPS level

1. Provides the definition of PRESCRIPTIVECPS (Section 3.1), according to which it is the system of interacting *physical* and *cyber* (sub-)systems. Here, the cyber system consists of inter-connected agents forming a hierarchy based on their obedience to each other (see Figure 1.6). Agents collectively manage instances of so-called *flexibility* (world), *decision*, and *prescription* models, which are short-lived, focus on the future, and represent (1) capability, an (2) (agent owner) intention, and (3) actions to change the behaviour (state) of a physical system, respectively.
2. Presents the definition of *flexibility*, *decision*, and *prescription* models, by showing how these models are related and formally defined (Section 3.2).
3. Defines the semantics of PRESCRIPTIVECPS in terms of core planning operations transforming *flexibility*, *decision*, and *prescription* model instances (Section 3.3).
4. Defines the set of roles an agent can potentially take depending on the set of core planning operations it supports (Section 3.5).
5. Exemplifies the defined concepts by elaborating the specialized PRESCRIPTIVECPS instance from the MIRABEL project (Section 3.6).

Thesis contributions at the agent level




6. Defines functional and non-functional requirements with regards to the model (and related data) management to be satisfied by a PRESCRIPTIVECPS agent (Section 3.7).
7. Proposes the architecture of an agent software system integrating the number of components specially designed or enhanced to satisfy the agent-specific functional and non-functional requirements (Section 4.1). Additionally, the overviews of capabilities and functionality of these components are provided.

Thesis contributions at the agent component level

8.  Proposes and generalizes (for other PRESCRIPTIVECPS instances) a *multi-dimensional data warehouse* schema (Chapter 5) to be used for storing flexibility and prescription model instances (and related data) in the MIRABEL use-case. The schema is unified as it can be used by different CPS agent roles in

the MIRABEL use-case. Alternative data modelling strategies based on typical queries from the energy domain were discussed and experimentally evaluated.

9. ■ Proposes techniques to *aggregate* flexibility model instances and *disaggregate* prescription model instances and demonstrates these techniques using the MIRABEL use-case (Chapter 6). The (lossy) aggregation and (exact) disaggregation is vitally important for the large-scale instances of PRESCRIPTIVECPS (such as the one from the MIRABEL use-case), as the aggregation and disaggregation allow (substantially) reducing the complexity of decision problems while still allowing to reproduce valid prescriptions at lower aggregation levels. The decision problems of the manageable size are required for (near) real-time planning where time for decision making is often limited and (good enough) decisions need to be generated in a limited time. The thesis also presents an efficient incremental aggregation technique allowing to update aggregated model instances incrementally (after changes in input), requiring no complete re-computation.
10. ▣ Proposes the design of a DBMS with built-in time series *forecasting* functionality (Section 7.1), which offers increased usability, productivity, and performance compared to the traditional forecasting process. The DBMS supports so-called *forecast queries* allowing to compute forecasted (future) time series values inside a DBMS based on historical measurements from a database. The thesis discusses integration aspects, presents the requirements and challenges of *time series forecasting*, and proposes an ANSI/SPARC-based architecture allowing a transparent, efficient, and end-to-end execution of forecast queries.
11. ▣ Demonstrates domain-specific (MIRABEL-specific) approaches and software architecture for *forecasting* instances of flexibility models (Section 7.2) – entities that are far more complex than time series and therefore cannot be directly forecasted using the DBMS with built-in time series forecasting (Section 7.1). The demonstrated approaches convert (augment) time series into instances of a flexibility model utilizing detailed information about a domain.
12. ■ Proposes the design of a DBMS with built-in *optimization* problem solving functionality (Chapter 8), which allows solving optimization (decision) problems “inside” a DBMS. The DBMS enables so-called *solve queries* that offer a common language for queries and optimization models, both expressed using Structural Query Language (SQL) constructs. As part the contribution, the thesis presents an extensible solver infrastructure allowing to integrate (into a DBMS) a variety of solvers, each of which tackles a specific class of problems, e.g., *linear programming*. Additionally, the thesis presents query optimization techniques allowing to increase the execution performance and/or result quality. The presented results of extensive experiments with the PostgreSQL-based implementation witness both significantly better user productivity and problem solving performance.

13.  Demonstrates simplified powerful ways to analyse past, current, and future data based on so-called hypothetical *what-if scenarios* and flexibility and prescription models from the MIRABEL use-case (Chapter 9). These what-if scenarios, as we elaborate in this thesis, can be expressed in SQL and natively supported by the agent's DBMS. When the DBMS additionally features built-in forecasting and optimization functionalities, an analyst can be offered an integrated prescriptive analytics platform supporting most of the tasks of planning (see Figure 1.3).
14.  Proposes a real-time data management architecture for processing instances of flexibility and prescription models under (soft or hard) timing constraints (Chapter 10). The architecture enables *storage* and *query processing* optimizations and offers many useful features such as *model-specific*, *approximate*, and *subscription-based* queries. Experiments with our initial MIRABEL-specific prototype, integrating (some of) our proposed techniques, witness significant performance improvements with respect to (historical) measurement storage, flexibility model aggregation, and forecast model maintenance.
15.  Presents the design of a *graphical user interface* (GUI) for displaying and visually analysing instances of flexibility and prescription models from the MIRABEL use-case (Chapter 11). The presented initial implementation is able to visualize large amounts of such complex model instances and offer generalized and in-depth analysis of all such data.

Summary of contribution

All these thesis contributions can be summarized as follows:

The thesis provides:

- The **model of a CPS** with the capabilities of mixed-initiative (human- and machine-based) distributed continual planning and control.
- The design and experimental evaluation of the following **prescriptive analytics techniques**, all contributing the process of planning (Figure 1.3):
 - planning-specific techniques allowing to effectively **generate**, **forecast**, **aggregate**, and **disaggregate** complex (world) models of a physical system.
 - data management and analysis techniques enabling **(near) real-time processing** and new ways of and **querying** and **analysing** data about *past*, *presence*, and/or *future*.

- The **use-case** from the smart-grid domain, showing how the introduced concepts are applicable in real-world.

1.7 Thesis Overview

The rest of the thesis follows the structure shown in Figure 1.7. As denoted by indentation and grouping in the figure, the chapters of this thesis focus on different levels of the PRESCRIPTIVECPS scale (i.e., the level of a CPS, an agent software, or an agent software component) and have different aims, summarized below.

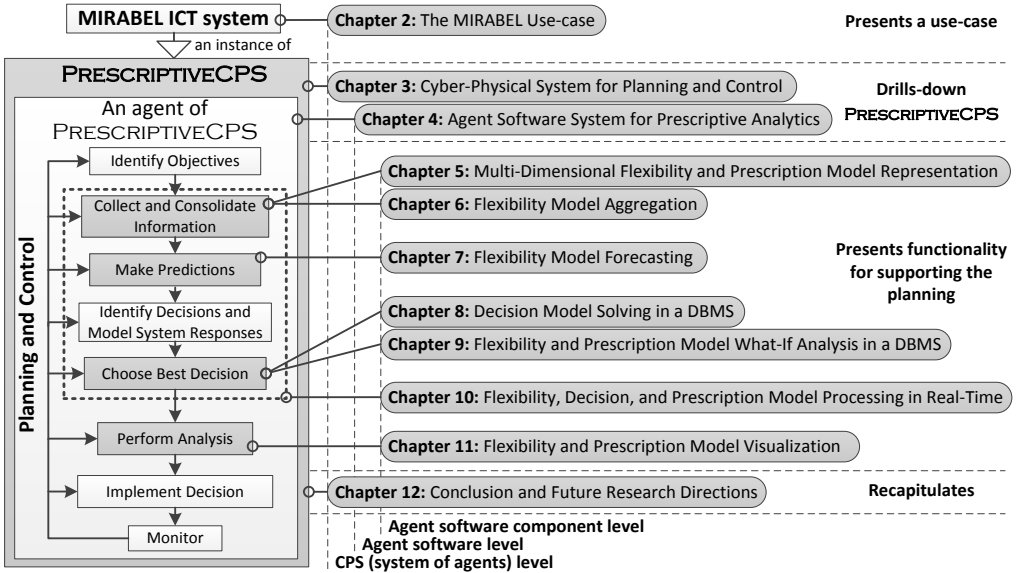


Figure 1.7: The structure of the thesis

Present a use-case Chapter 2 presents the real-world example of a large-scale CPS with planning and control capabilities. The CPS takes and realizes decisions continuously and automatically to balance electricity consumption and production in the power-grid with a substantial amount of *renewable energy sources* (RES). As shown later, this CPS is an instance of a generic PRESCRIPTIVECPS.

Drill-down PrescriptiveCPS Chapter 3 elaborates PRESCRIPTIVECPS at full scale by discussing its overall architecture, entities managed by the system, agent roles, core functionality, etc. Chapter 4 drills down to the level of a CPS agent software and presents its architecture and core functionality. The presented architecture is generic as it can be employed by PRESCRIPTIVECPS agents irrespectively to their roles.

Present functionality for supporting the planning After presenting the software architecture, the thesis drills down to the agent's software component level and follows the process of planning (see Figure 1.7) while presenting individual components and/or their functionality. Specifically, Chapter 5 follows the MIRABEL use-case and elaborates the storage of agent's data using a multi-dimensional data warehouse. Chapter 6 also follows the MIRABEL use-case and present techniques to aggregated complex flexibility model instances. Chapter 7 presents domain-specific and generic built-in approaches to forecast flexibility model instances (e.g., time series). Chapter 8 presents built-in approach to solve optimisation problems. Chapter 9 demonstrates the use of what-if analysis using an integrated DBMS in the MIRABEL use-case. Chapter 10 considers planning under the presence of (soft or hard) constraints on processing time. Finally, Chapter 11 considers the visual representation and analysis of complex flexibility and prescription models used in planning.

Recapitulate Chapter 12 summarizes the contributions, concludes the thesis, and presents future work.

Chapter 2

The MIRABEL Use-case

In this chapter, we present a large-scale hierarchical ICT system that will be used throughout the thesis as the use-case example of a CPS with planning and control capabilities. The ICT system was designed and prototyped in the MIRABEL (abbr. Micro-Request-Based Aggregation, Forecasting and Scheduling of Energy Demand, Supply and Distribution) project [25] from the smart-grid domain. The system automatically takes and realizes decisions to balance electricity consumption and production in the power-grid with a substantial amount of *renewable energy sources* (RES). The content of this chapter is based on Publication [1].

2.1 MIRABEL Project Overview

Today, more and more renewable energy sources (RES) such as windmills are added to the European electricity production portfolio since they offer sustainable energy with a much lower environmental impact compared to the conventional energy sources. However, due to the intermittent nature of the RES supply, the balancing of energy in the power grid with the substantial amount of RES (>30% of a total production) becomes a really challenging task, as it is not always possible to store larger amounts of energy [26]. Hence, the capacity of RES is not always fully utilized during nights when most consumers are inactive, and there is often not enough supply from RES during days when most consumers are active. To address this challenge (among others), the European electricity grid is being incrementally transformed into a *smart-grid* – which is a new generation power grid that utilizes a modern information and communication technology (ICT) to produce, distribute, and consume energy in a more intelligent way.

MIRABEL (abbr. Micro-Request-Based Aggregation, Forecasting and Scheduling of Energy Demand, Supply and Distribution) is an EU FP7 research project [25] that develops and prototypes a multi-agent ICT system [1] for the European smart-grid. The

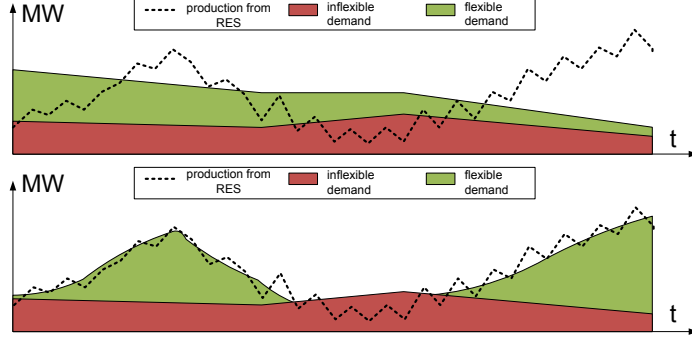


Figure 2.1: The example of loads before and after the MIRABEL ICT system balances demand and supply in the power grid

MIRABEL system opens the possibility for millions of individual energy consumers to consume more energy at times when there is excessive supply from RES and less energy when a RES supply is insufficient, as well as provides convenient ways to balance the grid (see Figure 2.1) for the actors (legal entities) of the European electricity market [27].

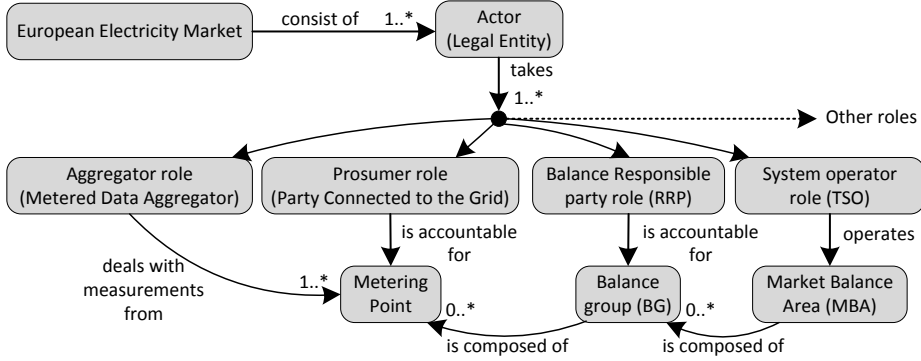


Figure 2.2: The relevant concepts of the European electricity market

In the European electricity market (see Figure 2.2), actors in the roles of *Balance Responsible Party* (BRP) and *Transmission System Operator* (TSO), as defined by the Harmonized Electricity Market Role Model [27], concerns balancing demand and supply in the grid and, therefore, benefit from the MIRABEL ICT system. Here, an actor in the *TSO* role, among other activities, handles imbalances at the level of *market bal-*

ance area (MBA), which is a sub-domain within the complete European power-grid. To ensure the balance, the TSO actor meters the difference between consumption and production at MBA level in real-time and triggers interventions to keep energy consumption and production balanced (in sync) at every second. Similarly, an actor in the *BRP* role is financially accountable for imbalances in a so-called *balance group* (BG), which is a sub-domain within an MBA. The BRP actor manages the loads of *prosumers* (e.g., entities that consume and/or produce energy) and aims to reduce imbalances, all these prosumers collectively incur in a balance group while consuming and producing electricity. For actors in the *TSO*, *BRP*, and *prosumer* roles, the MIRABEL ICT system brings various advantages such as (1) savings and earnings on electricity, (2) reduced imbalances and peak-loads, (4) higher RES integration, (5) CO_2 reductions, etc.

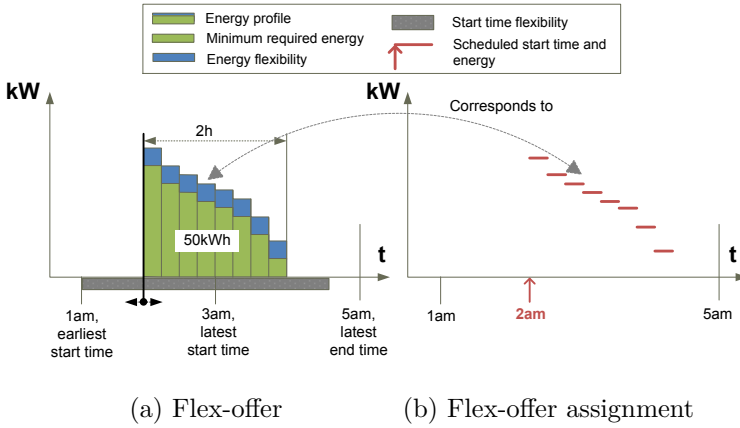


Figure 2.3: Structural elements of a flex-offer (a) and a corresponding flex-offer assignment (b)

MIRABEL relies on the concept of a *flex-offer* – an entity that explicitly captures a prosumer’s capability to consume (or produce) energy in a flexible manner. Each flex-offer captures elements depicted in Figure 2.3a. Among other elements, a flex-offer defines an energy profile with a number of slices, each of which specifies the minimum and maximum amounts of energy to be consumed (or produced) at a particular time interval, which is relative to the starting time of an appliance (e.g., a dishwasher). According to the flex-offer, an appliance can be scheduled (by a BRP) to started at any point in time within a bounded time interval (see *start time flexibility*). The scheduled flex-offer is termed a *flex-offer assignment*, and it is a *time series* (i.e., flex-offers with empty time and energy flexibility bounds) defining an actual starting time of an appliance (see *start time*) and concrete amounts to be consumed (or produced) at each time slice, which are specified by a flex-offer profile (see Figure 2.3b). We will elaborate later how such flex-offers and flex-offer assignments are leveraged in MIRABEL.

2.2 MIRABEL ICT System

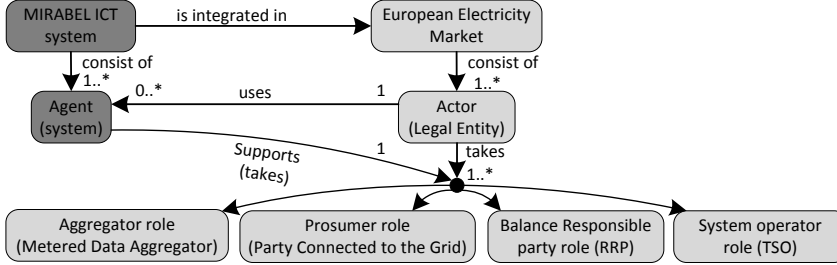


Figure 2.4: The relevant concepts of the MIRABEL ICT system and the European electricity market

The MIRABEL ICT system consists of a large number of agents used by the actors of the European electricity market (Figure 2.4). An agent supports a particular role of an actor, which we denote by associating a role to an agent, e.g., a *BRP agent* or an *agent in the BRP role*. The agents of the MIRABEL ICT system form the multi-level hierarchy shown in Figure 2.5. The levels four to one of the hierarchy include the agents in the *TSO*, *BRP*, *aggregator*, and *prosumer* roles, respectively.

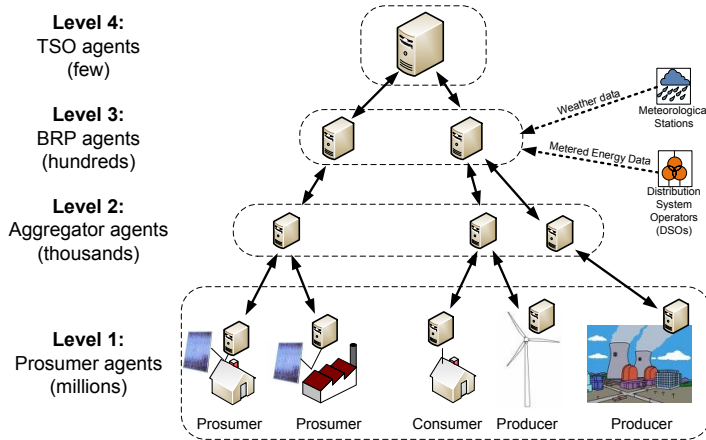


Figure 2.5: Agents and their hierarchical organization in the MIRABEL ICT system

The agents of the MIRABEL ICT system collectively realize the workflow, shown

in Figure 2.6. We now describe these agents and elaborate flex-offer and flex-offer assignment management activities they are involved in.

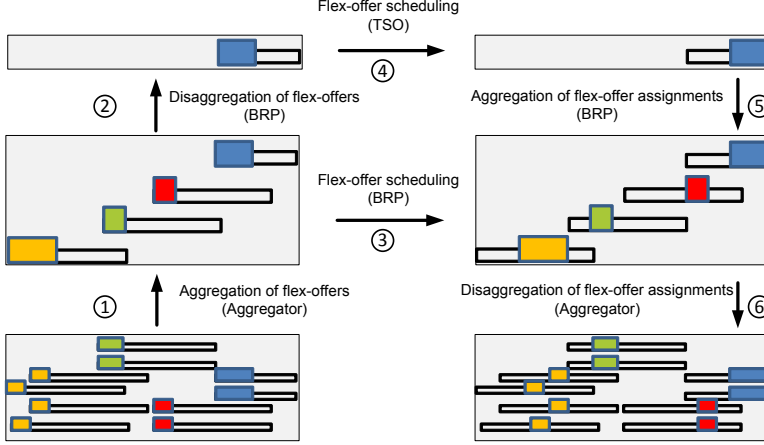


Figure 2.6: The workflow of flex-offers and flex-offer assignments in the MIRABEL ICT system

A *prosumer agent* generates a flex-offer for every intent of the prosumer to consume or produce electricity in a near future, e.g., in the day ahead. For each flex-offer, the agent receives the corresponding flex-offer assignment respecting the constraints specified in a flex-offer and indicating actual time and the amounts of load that has to be consumed or produced. The details of flex-offer generation and flex-offer assignment processing will be elaborated in Section 2.3.

An *aggregator agent* aggregates N (micro) flex-offers into M (macro) flex-offers and disaggregates corresponding (macro) M flex-offer assignments into N (micro) flex-offer assignments such that $M \leq N$ (see ① and ⑥ in Figure 2.6). By aggregation, the agent abstracts (micro) energy loads from individual households with the (macro) loads from individual feeders, transformers, or radials of the distribution network for the simplified handling of loads (flex-offers) at the BRP level.

A *BRP agent* plans energy for a certain horizon ahead in time (e.g., day-ahead) to minimize imbalances between consumption and production in a balance group. In planning, both *flexible* and *inflexible* loads of consumption and production are considered. Here, flexible loads are defined as (macro) flex-offers that are collected from aggregator agents. The inflexible loads of production (e.g., supply from RES) and consumption (e.g., inflexible demand of consumers) are defined as *time series* and are forecasted based on external data (e.g., metered energy, wind speed) from external data sources such as *distribution system operations* (DSOs – aka. grid companies) and meteorological sta-

tions. The planning results in a number of flex-offer assignments (see ③ in Figure 2.6). Furthermore, depending on the BRP's business strategy, the BRP allocates some part of its flexible loads for the purpose of energy balancing at the grid (inter-balance group) level (see ② in Figure 2.6). In the form of flex-offers, the BRP agent makes these loads available to a TSO agent. When scheduled TSO loads in the form of flex-offer assignments are received by the BRP agent, they are aggregated with other BRP's flex-offer assignments to be realized by underlying prosumer agents, via the aggregator agent (see ⑥ in Figure 2.6).

A *TSO agent* utilizes flexible BRP loads (specified as flex-offers), among other types of reserve loads, to stabilize the grid in the presence of grid-level (the MBA level) imbalances (see ④ in Figure 2.6).

2.3 Example of Electric Vehicle Charging

We now demonstrate the typical flow of events in the MIRABEL ICT system by employing the use-case example of charging an electric vehicle (EV). The use-case involves a prosumer, an aggregator, a BRP, and a TSO agent.

1. An EV owner comes home at 11pm and wishes to recharge the EV's battery at the lowest possible price before the next morning. Once plugged in, the outlet recognizes the EV and chooses a default energy consumption profile, according to which the minimum required energy is 50kWh and the default charging completion time is 5am.
2. The prosumer agent automatically generates a flex-offer (as in Figure 2.3) and sends it to the aggregator agent, where it is aggregated with other similar flex-offers (see ① in Figure 2.6). The aggregator agent sends aggregated flex-offers to the BRP agent.
3. The BRP agent schedules these (aggregated) flex-offers, while taking into account weather and inflexible energy forecasts (see ③ in Figure 2.6). When it is done, the corresponding (aggregated) flex-offer assignments are sent back to the aggregator agent. Simultaneously, the BRP agent disaggregates some of the (aggregated) flex-offers (see ② in Figure 2.6) and sends them to the TSO agent for the grid-level balancing.
4. The prosumer agent receives a flex-offer assignment (corresponding to the original flex-offer) from the aggregator agent after it disaggregates its (aggregated) flex-offer assignments (see ⑥ in Figure 2.6). According to the flex-offer assignment, the EV's battery is scheduled to start charging at 2am, as such scheduling lowers an energy demand peak at 0am and consumes surplus production of RES at 2am at the BRP (BG) level.

5. At 2am, the TSO notices a shortfall of supply at the grid level and instructs the BRP agent to shift the specific amount of load from 2am to 3am (see 4 in Figure 2.6). The BRP agent reschedules its flex-offers correspondingly (see ⑤ in Figure 2.6) and delivers new (aggregated) flex-offer assignments to the aggregator agent.
6. The prosumer agent receives a new flex-offer assignment (corresponding to the original flex-offer) from the aggregator agent after it disaggregates the new flex-offer assignments from the BRP agent (see ⑥ in Figure 2.6). According to the flex-offer assignment, the charging of the EV's battery is now scheduled to start at 3am instead at 2am.
7. The prosumer agent starts charging the EV's battery at 3am and finishes the charging at 5am. Metered consumption loads incurred by the charging of the EV are supplied to the BRP agent, e.g., via a local DSO (out of scope of this thesis).
8. The next month, the BRP sends an energy bill to the consumer. The bill reflects the reduced energy cost in return for user's offered flexibility.

2.4 Summary

In this chapter, we presented the multi-agent, multi-role, and multi-level ICT system from the smart-grid domain. The system was designed and prototyped in the MIRABEL project to balance electricity consumption and production in the power grid with the substantial amount of renewable energy sources (RES). To balance the grid, and thus provide various benefits to stakeholders, the system exploits flexibilities offered by a large number of prosumers (e.g., households) in the representation of flex-offers.

In the next chapter, we generalize the MIRABEL ICT system and the concepts proposed in the MIRABEL project. As a result, we present a so-called PRESCRIPTIVECPS, which is a generic CPS with automated planning and control capabilities.

Chapter 3

Cyber-Physical System for Planning and Control

We now present a PRESCRIPTIVECPS, which is our proposed (conceptual model of a) CPS for mixed-initiative (human- and machine-based) distributed continual planning (DCP). PRESCRIPTIVECPS consists of two interacting systems: a *cyber system* and *physical system*. Here, the cyber system continuously and automatically (1) monitors the state of the physical system, (2) infers the system's responses to actuations, (3) makes decisions to achieve desired goals, and, finally, (4) controls the physical system to follow the decisions. In this chapter, we elaborate (1) the architecture of PRESCRIPTIVECPS, (2) the types of models managed by PRESCRIPTIVECPS, (3) core operations used to manage these models, and (4) the roles of PRESCRIPTIVECPS agents. Additionally, we show that the MIRABEL ICT system is the specialized instance of PRESCRIPTIVECPS. Finally, we conclude the chapter by providing a number of common model (and other data) management requirements that need to be addressed for the MIRABEL's, as well as other similar instances of PRESCRIPTIVECPS.

3.1 Definition of PrescriptiveCPS

Assume a *physical system* P consisting of a number of individual physical sub-systems $\{P_1, P_2, \dots, P_m\}$, the performances of which influence or determine the overall performance of P . Let S be a *cyber system* consisting of the set of autonomous agents, $S = \{a_1, a_2, \dots, a_n\}$, having an electro-mechanical links to (some or all) physical sub-systems and aiming to plan and control the performance of the physical system P (or $\subseteq P$). For this intent, the agents are networked and communicate according to a hierarchy defined by the binary relation \leq partially ordering the agents based on their obedience to each

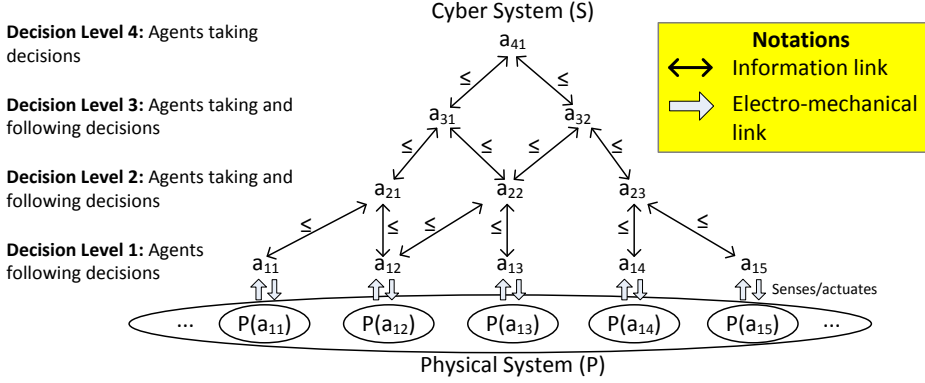


Figure 3.1: The hierarchical organisation of agents in PRESCRIPTIVECPS

other (see Figure 3.1). The notation $a_{21} \leq a_{31}$ represents the fact that a_{21} can derive its individual plan based on its own objectives, but the plan must be consistent with the “master plan” prescribed by a_{31} . According to the hierarchy, an agent may have multiple (immediate) *successor agents*, denoted as *masters*($a \in S$), decisions of which the agent has to follow. Similarly, an agent may have multiple (immediate) *predecessor agents*, denoted as *children*($a \in S$) to which the agent prescribes decisions. With respect to the hierarchy, an agent either (1) *takes a decision* if it has no successor agent, (2) *follows a decision* if it has no predecessor agent, (3) *takes and follows decisions* if it has both successor and predecessor agents. The coupling and the interaction of these two systems (P and S) are summarized in the following (conceptual) definition of PRESCRIPTIVECPS:

Definition 3.1. PRESCRIPTIVECPS is a system (CPS) consisting of electro-mechanically coupled cyber and physical systems, where the agents of the cyber system (S) are organized hierarchically based on obedience to each other to optimally manage the performance of the physical system (P).

The model of a PRESCRIPTIVECPS agent is shown in Figure 3.2. According to the model, the agent $a_x \in S$ is a hardware- and software-based computer system with a number of links for interacting with (1) a physical system, (2) human users, (3) external data sources, and (4) successor and predecessor agents. The agent has the capability to directly sense and actuate a physical (sub-)system $P(a_x) \subseteq P$ (see the

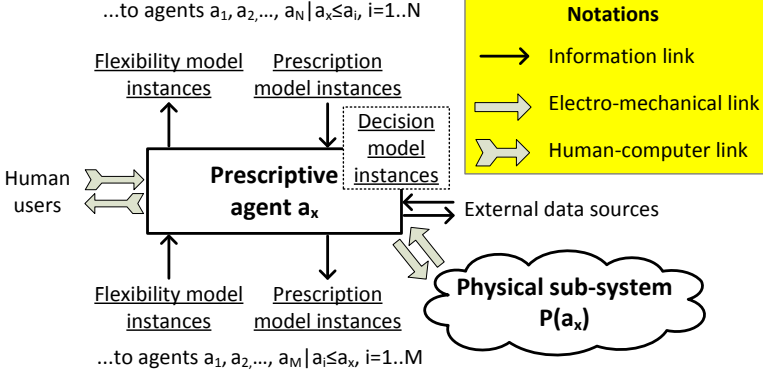


Figure 3.2: The model of a PRESCRIPTIVECPS agent

electro-mechanical link). It also allows human operators and administrators to monitor and program the behaviour of the agent (see the *human-computer link*). Additionally, the agent has an access (via the information link) to external data sources (see the left-right *information link*) for accessing external data (e.g., *weather forecasts*) to be used in planning. Finally, the agent communicates with the immediate successor and predecessor agents ($masters(a_x)$ and $children(a_x)$) by continuously exchanging so-called *flexibility model* and *prescription model* instances (see the top-down *information link*). Internally, the agent builds so-called *decision model* instances to produce *prescription model* instances from *flexibility model* instances. As we show later, depending on the set of roles the agent realizes, some of these links and models might not be integrated or managed by an agent.

The *flexibility model*, the *decision model*, and the *prescription model* are special models, representing a (1) capability, (2) an intention, and (3) actions to change the behaviour (state) of a physical system (e.g., $P(a_x)$), respectively. We now elaborate these models and describe the core operations an agent uses to manipulate instances of these models.

3.2 Models of a Physical System

We assume the common notion of a model, where the *model* is a generic entity (structure) that must be instantiated to a *model instance* [28] before it can be used for its intended purpose. Following this notion, an agent of our proposed PRESCRIPTIVECPS continuously maintains *flexibility*, *decision*, and *prescription* model instances, which are

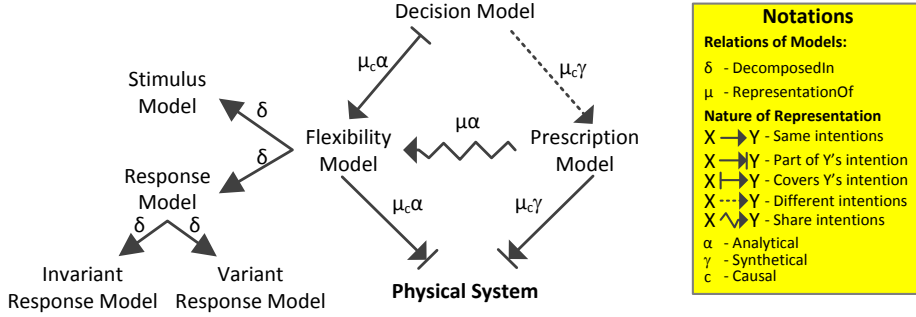


Figure 3.3: The relationships and decompositions of models managed by PRESCRIPTIVECPS

short-lived, focus on the future, and either describe (represent) or specify (prescribe) certain aspects of the physical system at a particular moment in time. The relationships between our proposed models and their decompositions are visualized in Figure 3.3, where the common model modelling notations [29,30] are used. Note, for simplicity, we use the term *model* to denote a *model instance* where it is clear.

Flexibility Model

A *flexibility model* is the special (but still general enough) *world model* [31]. An instance of this model explicitly defines how a physical system can be stimulated and how it responds to every possible stimulus at a particular moment in time. In other words, it defines *stimuli* (*Stimulus Model*) and *responses* (*Response Model*) describing a capability to change the physical system's state (aka. flexibility) together with properties and the behaviour of a physical system under various conditions and alternative stimulus (actuations) applied to the physical system. These are described from the perspective of some specific time moment (e.g., current time) and, therefore, an instance of the flexibility model is valid as long as the physical system exhibits the represented stimulus-response behaviour (causal representation). The structure and the properties of a *flexibility model* are elaborated in the following definition:

Definition 3.2. A flexibility model f is (or can be defined as) a 5-tuple (V, S, R, D^V, D^R) . Here, V is a set of so-called prescription variables defining the aspects of a physical system an agent has the capability to prescribe (e.g., power output, voltage level, flow rate, or starting time). S is a so-called a stimulus rela-

tion capturing value combinations the prescription variables can take. The relation consists of a set of tuples $(d_1, d_2, \dots, d_{|V|})$ where each tuple in the set is called a stimulus. Each d_n is a member of the domain D_n^V , where $D^V = D_1^V \times D_2^V \times \dots \times D_n^V$ is termed the stimulus domain, and $S \subseteq D^V$. $R : D^V \rightarrow D^R$ is a so-called response function defining physical system responses to stimuli. Here, a response $r = R(s)$ is such that $r \in D^R$, where D^R is called the response domain. When R is defined as a composition of two functions $R_i() \oplus R_v(s)$ such that $R_i : \emptyset \rightarrow D^R$, $R_v : D^V \rightarrow D^R$, and the pair (D^R, \oplus) forms an Abelian group, then we say that the response R has invariant and variant parts R_i and R_v , respectively.

We now provide two examples of a flexibility model. The flexibility model in the first example takes the 5-tuple representation. The flexibility model in the second example takes a different representation, but can easily be transformed into the 5-tuple representation.

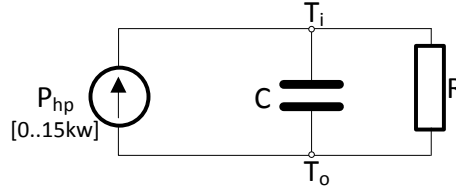


Figure 3.4: The temperature model of a simple household with a heat pump

Example 3.1 (A household's temperature flexibility model). A flexibility model f_{house} can represent the expected future temperatures of a household with an electrical heat pump (see Figure 3.4). Suppose that the expected temperature of the household follows $T(t) = P_{hp}R(1 - e^{-\frac{t}{RC}}) + (T(0) - T_o)e^{-\frac{t}{RC}} + T_o$, where P_{hp} is the power delivered by the heat pump, $T(0)$ is the initial temperature in the household, T_o is the temperature outside the household, and C and R are household's thermal capacity and heat resistance, respectively. Suppose that power output of the heat pump (P_{hp}) can be adjusted and set to any power value in the range between 0kW to 10kW. Furthermore, suppose that at the current moment in time, $T_o = -5^\circ\text{C}$, $T(0) = 15^\circ\text{C}$, $C = 0.5\text{kWh}/^\circ\text{C}$, and $R = 0.1^\circ\text{C}/\text{kW}$. Then, the corresponding household's flexibility model f_{house} can be defined as $(\{P_{hp}\}, \{P_{hp} | 0 \leq P_{hp} \leq 15\}, (P_{hp}, t) \mapsto 0.1P_{hp}(1 - e^{-\frac{t}{0.05}}) + 20e^{-\frac{t}{0.05}} - 5, \mathbb{R}, \mathbb{R} \times \mathbb{R})$. Here, the functions $t \mapsto 20e^{-\frac{t}{0.05}} - 5$ and $(P_{hp}, t) \mapsto 0.1P_{hp}(1 - e^{-\frac{t}{0.05}})$ are the

invariant and variant responses, respectively. Unlike the variant response, the invariant response is not dependent on the stimulus (P_{hp}).

Example 3.2 (A flex-offer as a flexibility model). A flex-offer, as defined in Section 2.1, is a flexibility model which defines flexible loads of a single consumption and/or production unit (e.g., an electric vehicle or a heat pump). A flex-offer can be transformed into the 5-tuple representation $f_{unit} = (V, S, R, D^V, D^R)$, where V includes starting time (of a unit) and energy amounts for every slice in the profile as prescription variables (see Figure 2.3a). The stimulus relation S includes all possible flex-offer assignments (see Figure 2.3b) that can be prescribed according to the flex-offer. The response model R is the identity function $R: x \mapsto x$, which means that stimuli (flex-offer assignments) fully define responses.

Prescription Model

A *prescription model* represents one of the alternative stimuli defined by a flexibility model. In other words, it describes how a physical system has to be stimulated at a particular moment in time to, for example, achieve some desired objectives. The properties of the prescription model are elaborated in the following definition:

Definition 3.3. A prescription model (or *prescription* for short) is an entity p conforming to the flexibility model $f = (V, S, R, D^V, D^R)$ such that $p \in S$ ($S \subseteq D^V$).

We now provide two prescription model examples that are based on the previously presented flexibility models (f_{house} and f_{unit}).

Example 3.3 (Household's temperature prescription model). With respect to the described flexibility model f_{house} , the heater's power output $P_{hp} = 10kW$ is a valid prescription model.

Example 3.4 (A flex-offer assignment as a prescription model). A flex-offer assignment, as defined in Section 2.1, is a prescription model which defines a starting time of an appliance (scheduled start time, aka. t_{start}) and concrete energy amounts to be consumed (or produced) at each time slice specified in a flex-offer profile.

Decision Model

A *decision model* is a mathematical model representing a decision (optimization) problem. It is derived from the flexibility model by decomposing it into entities such as numbers, variables, sets, equations, functions and operators. Together with additional objectives and constraints that are specific to an agent, these define an optimization problem, the solution of which includes a prescription leading the physical system to a state desired by an agent.

Definition 3.4. A decision model is the following mathematical model derived from a flexibility model $f = (V, S, R, D^V, D^R)$:

$$\begin{aligned} \underset{X}{\text{Minimize:}} \quad & f_1(X), f_2(X), \dots, f_k(X) \\ \text{Subject To:} \quad & g_i(X) \leq 0, i = 1, 2, \dots, m \\ & h_j(X) = 0, j = 1, 2, \dots, p, \end{aligned} \tag{3.1}$$

Here, X is a set of decision variables that include prescription variables, i.e., $X \supseteq V$. $f_1(X), \dots, f_k(X)$ are objective functions with co-domains in \mathbb{R} . The $g_i(X) \leq 0$ and $h_j(X) = 0$, $i = 1 : m$, $j = 1 : p$, are inequality and equality constraints, respectively.

An objective function(-s) might potentially be derived from the response model (R) if an agent's intention is to minimize or maximize some objective (e.g., a profit) dependent on a response. Similarly, the constraint (in-)equalities might potentially be derived from the stimulus model (S) in order to bound the prescription variables to the values of valid stimuli, i.e., $(x_1, x_2, \dots, x_{|V|}) \in S$ where $x_i \in V \subseteq X, \forall i = 1..|V|$. Finally, as we will discuss later, prescriptions from the successor agents might be incorporated into the objective function or (in-)equalities to be able to meet the “master plan” from the successor agents. We now provide two decision model examples that are based on the previously presented flexibility models (f_{house} and f_{unit}).

Example 3.5 (Household's temperature decision model). To stabilize the household's temperature at 20°C by prescribing the heat pump's power level P_{hp} , the following decision model d_{hp} is defined based on the flexibility model $f_{house} = (\{P_{hp}\}, S, R, D^V, D^R)$:

$$\begin{aligned} \underset{P_{hp}}{\text{Minimize:}} \quad & (R(P_{hp}, +\infty) - 20)^2 \\ \text{Subject To:} \quad & P_{hp} \in S \end{aligned} \tag{3.2}$$

Based on the flexibility model f_{house} , decision models can be built for other types of prescription objectives, e.g., to minimize the energy P_{hp} while keeping the temperature at least 20°C in the household.

Example 3.6 (A flex-offer scheduling decision model). The following decision model defines the problem of balancing the consumption and production from a single flexible consumption unit (f_{unit}) and inflexible RESs during the 24 hour time period:

$$\begin{aligned}
 &\text{Minimize:} && \sum_{t=1h}^{24h} |RES(t) - f_a(t)| \\
 &\text{Subject To:} && t_{\text{start}} \in \text{starttime_bound}(f_{\text{unit}}) \\
 & && e_i \in \text{energy_bound}(f_{\text{unit}}, i), i = 1..k \\
 & && f_a = \text{generate_assignment}(f_{\text{unit}}, t_{\text{start}}, e_1, e_2, \dots, e_k)
 \end{aligned} \tag{3.3}$$

Here, $t_{\text{start}}, e_1, e_2, \dots, e_k$ are the starting time and energy amounts at every slice in a profile (see Figure 2.3b). The `starttime_bound` and `energy_bound` are functions returning starting time and energy bounds as defined by a flex-offer. The `RES` is a time series of RES production, and `generate_assignment` generates a flex-offer assignment (time series) based on the supplied flex-offer and the prescription variables ($t_{\text{start}}, e_1, e_2, \dots, e_k$).

As seen from the examples above, *flexibility*, *decision*, and *prescription* models represent a (1) capability, an (2) (actor's) intention, and (3) actions to change the behaviour (state) of some physical (sub-)system $P' \subseteq P$ at some moment in time. We denoted physical systems underlying a flexibility model f , a decision model d , and a prescription model p as $P(f)$, $P(d)$, and $P(p)$, respectively. Note, these should be distinguished from the physical system $P(a_x)$ an agent a_x has a direct electro-mechanical link with.

To summarize, a flexibility model is a special “world model” defining stimuli and responses of a physical system. The decision model defines an optimization problem, a solution of which leads to a prescription. The prescription is a short-term plan leading into a desired state of the physical system. We now define the number of core planning operations used by an agent to deal with these three types of models.

3.3 Core Planning Operations with Flexibility, Decision, and Prescription Models

To deal with instances of flexibility, decision, and prescription models, an agent $a_x \in S$ relies on some (or all) of the planning operations, summarized in Table 3.1. We now elaborate each of them, grouped by the model type:

Table 3.1: The summary of planning operations with flexibility, decision, and prescription models

Flexibility model	Decision model	Prescription model
$build() \mapsto f$	$solve(f) \mapsto p_f$	$apply(p) \mapsto \emptyset$
$reduce(f_1, \dots, f_n) \mapsto f_a$	$solve_R(f, p_1, \dots, p_m) \mapsto p_f$	$reduce(p_1, \dots, p_m) \mapsto p_a$
$map(f_a) \mapsto (f_1, \dots, f_m)$	$solve_M(f_a, p_{f_a}) \mapsto (p_1, \dots, p_n)$	$map(p_a) \mapsto (p_1, \dots, p_n)$

Flexibility Model Operations

- $build()$ creates an instance of the flexibility model. In practice, this can be achieved by (1) directly monitoring the stimulus-response behaviour of the physical system (like in MIRABEL), (2) inferring a behaviour using historical data and predictive analytics (see Section 1.3) or machine learning [32] techniques, (3) pre-defining the behaviour in built-in flexibility models, or (4) applying a combination of these techniques.
- $reduce(f_1, f_2, \dots, f_n)$ aggregates the set of (smaller) flexibility model instances $F = \{f_1, f_2, \dots, f_n\}$ into a single (larger) flexibility model instance f_a . This operation can be either *lossless* or *lossy* depending on if the number of prescription variables in the aggregated model $f_a = (V_a, S_a, R_a, D_a^V, D_a^R)$ is equal to the total number of prescription variables in the models f_1, f_2, \dots, f_n . We use the term *lossless aggregation* to denote the case where $|V_a| = \sum_{(V,S,R,D^V,D^R) \in F} |V|$ and the term *lossy aggregation* to denote the case where $|V_a| < \sum_{(V,S,R,D^V,D^R) \in F} |V|$. Note that f_1, f_2, \dots, f_n , and f_a can be instances of different flexibility models (and/or flexibility models in different representations).
- $map(f)$ disaggregates a (larger) flexibility model instance f into the set of (smaller) flexibility model instances $\{f_1, f_2, \dots, f_n\}$. Note that f_1, f_2, \dots, f_n , and f can be instances of different flexibility models (and/or flexibility models in different representations).

Example 3.7 (The flexibility model management in MIRABEL). *The flexibility model building, aggregation, and disaggregation has been demonstrated as a sequential bottom-up workflow in the MIRABEL use-case (see Sections 2.2–2.3), where flex-offers are used as flexibility model instances.*

Decision Model Operations

- $solve(f)$ builds and solves a decision model instance that results in a prescription p_f conforming to a flexibility model f . The decision model instance represents intentions (objectives) of a **single agent** only. Therefore, the operation is intended to be used by an agent taking (highest level) decisions (see Section 3.1).

- $solve_R(f, p_1, p_2, \dots, p_m)$ builds and solves a decision model instance that results in a prescription p_f conforming to the flexibility model f . The decision model instance includes the prescriptions p_1, p_2, \dots, p_m from successor agents ($masters(a_x)$). Consequently, the decision model represents the intentions of **multiple agents**, specifically the agents a_x and $masters(a_x)$. Therefore, the operation is intended to be used by an agent taking and following decisions (see Section 3.1).
- $solve_M(f_a, p_{f_a})$ builds and solves a decision model instance that results into multiple prescriptions p_1, p_2, \dots, p_n , conforming to flexibility models f_1, f_2, \dots, f_n , respectively, such that $reduce(f_1, f_2, \dots, f_n) = f_a$. The decision model instance represents the intention of a **single or multiple agents** (a_x or a_x and $masters(a_x)$) and aims to disaggregate the prescription p_{f_a} conforming to f_a into a number of individual prescriptions to be issued to predecessor agents ($children(a_x)$). The operation is intended to be used by an agent taking or both taking and following decisions (see Section 3.1).

The $solve$, $solve_R$, and $solve_M$ can be seen as composite functions in the pattern $eval(build(f, \dots))$. Here, the $build$ function transforms the instances of flexibility model into the instances of a decision model. It is carefully designed by humans to reflect agent-specific (external) objectives and constraints. The $eval$ function is a fixed (static) and invokes a respective solver to produce the instance(-s) of a prescription model from the instance of a decision model.

Example 3.8 (Decision model building and solving in MIRABEL). *In the MIRABEL use-case (see Chapter 2), TSO and BRP agents realize the $solve$ and $solve_R$ operations, respectively. The $solve_M$ operation is not considered in the use-case, but it can be potentially used by TSO, BRP, and aggregator agents to minimize incompatibilities of flex-offer assignments (prescriptions) at two adjacent levels (e.g., the BRP and aggregator levels). Such incompatibilities may occur when disaggregating flex-offer assignments under (micro and macro) flexibility model incompatibilities, which may occur, for example, due to network delays and rapidly changing flex-offers.*

Prescription Model Operations

- $apply(p)$ translates a prescription model instance p into a valid physical action, which might cause a physical system's state change. For example, this operation might translate the prescription $p = 5kW$ into (the set point of) continuous $5kW$ heater's power output, which is only changed when a new prescription is applied. This function realizes the *control* (as opposed to *planning*), introduced in Section 1.3.

- $reduce(p_1, p_2, \dots, p_m)$ aggregates a set of (smaller) prescription model instances $\{p_1, p_2, \dots, p_m\}$ into a (large) aggregated prescription model instance p_a conforming to a flexibility model instance f_a . It is a derived operation and can be formulated as $reduce(P) = solve_R(f_a, p_1, p_2, \dots, p_m)$. When a deterministic algorithm to transform p_1, p_2, \dots, p_m into p_a exists, a decision model (the use of $solve_R$) building and solving can be avoided.
- $map(p_a)$ disaggregates a (large) prescription model instance p_a conforming to f_a into the set of (smaller) prescription model instances $\{p_1, p_2, \dots, p_n\}$. It is a derived operation and can be formulated as $map(p_a) = solve_M(f_a, p_a)$. When a deterministic algorithm to transform p_a into $\{p_1, p_2, \dots, p_n\}$ exists, a decision model (the use of $solve_M$) building and solving can be avoided.

Example 3.9 (The prescription model management in MIRABEL). *The prescription model aggregation, disaggregation, and applying has been demonstrated as the sequential top-down workflow in the MIRABEL use-case (see Sections 2.2–2.3), where flex-offer assignments are used as prescription model instances.*

The set of planning operations is sufficient for systematically transforming momentary knowledge about physical system stimulus and response (encapsulated in flexibility models) into rigorous physical actions (potentially) leading a physical system into a desired state. The individual planning operations can be combined in many ways to realize various planning and control workflows, as shown in the next section.

3.4 Planning Workflow of PrescriptiveCPS

The described core planning operations can be nested into the numerous of ways, realizing various planning workflows in PRESCRIPTIVECPS. The following heat pump scheduling example presents one possible instance of such workflow:

Example 3.10 (Heat pump scheduling workflow). *Figure 3.5 shows the planning workflow of prescribing temperatures in three different households. It starts with the building of the flexibility models f_{house1} , f_{house2} , and f_{house3} (see build) representing the temperature responses of the three different households (see Example 3.1). Then, these models are aggregated (see reduce) into a “multi-household” flexibility model (f_{multi}) capturing the joint temperature response of the three households. When the aggregation is lossless, then f_{multi} is defined as $(V_1 \cup V_2 \cup V_3, S_1 \times S_2 \times S_3, (s_1, s_2, s_3) \mapsto (R_1(s_1), R_2(s_2), R_3(s_3)), D_1^V \times D_2^V \times D_3^V, D_1^R \times D_2^R \times D_3^R)$ where $f_i = (V_i, S_i, R_i, D_i^V, D_i^R)$, $i \in \{house1, house2, house3\}$. Then, the joint model f_{multi} is partitioned into f_{multi}^{C1} and f_{multi}^{C2} based on, for example, the energy companies (C1*

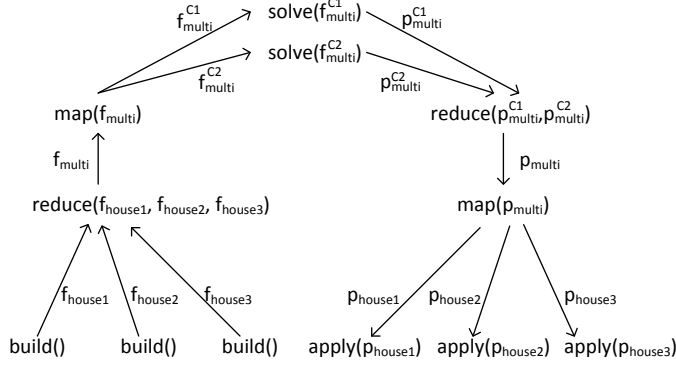


Figure 3.5: An example workflow (pursued by a single or multiple agents) to plan heat pump outputs at three different households

and $C2$) supplying energy to these households as these companies might have different power supply capabilities and energy planning interests. Suppose that each f_{multi}^{C1} and f_{multi}^{C2} represents the combined power responses of underlying heat pumps (rather than temperature responses) of a particular household group. An example of f_{multi}^{C1} or f_{multi}^{C2} is $(\{P_{total}\}, \{P_{total} | 0 \leq P_{total} \leq 1500\}, P_{total} \mapsto P_{total}, \mathbb{R}, \mathbb{R} \times \mathbb{R})$. During the decision making, energy companies decide energy amounts p_{multi}^{C1} and p_{multi}^{C2} to be consumed by each household group (see solve). These amounts are disaggregated into p_{multi} so that each household within the group gets, for example, equal temperatures in long-term, assuming that such objective is programmed in reduce. The prescription p_{multi} is mapped into the individual prescriptions p_{house1} , p_{house2} , and p_{house3} , specifying actual heat pump power outputs for each household (see map). Finally, p_{house1} , p_{house2} , and p_{house3} are used to program and control the behaviour of the heat pumps (see apply).

We have demonstrated one possible planning workflow specific to the heat-pump application. Various concurrent workflows with interleaved planning operations and nested *reduce* and *map* operations can be imagined and customized to fit organizational structures of the physical world in other applications (e.g., MIRABEL).

PRESCRIPTIVECPS sets no restriction on the set of planning operations a particular agent may support. In fact, a particular planning workflow can be realised by a *single*, *multiple*, or *all* collaborating agents of PRESCRIPTIVECPS. Irrespective to a concrete planning workflow and the number of agents realising it, PRESCRIPTIVECPS has the

semantics expressed in the following endless loop:

```

while true do
  |  $(f_1, f_2, \dots, f_m) \leftarrow (build_1(), build_2(), \dots, build_m());$ 
  |  $(p_1, p_2, \dots, p_n) \leftarrow plan(f_1, f_2, \dots, f_m);$ 
  |  $apply_1(p_1); apply_2(p_2); \dots; apply_n(p_n);$ 
end

```

Algorithm 1: The loop of PRESCRIPTIVECPS semantics

As shown in the algorithm, PRESCRIPTIVECPS continuously (1) *builds* flexibility model instances (using $build_i$, $i = 1..m$), (2) *transforms* flexibility model instances into prescription model instances using a concrete planning workflow ($plan$), and (3) *applies* prescription model instances to the physical world (using $apply_j$, $j = 1..n$). These actions must be performed sufficiently fast (e.g., by *parallelizing* them) to be able to effectively detect problems and take new opportunities in the physical world (see *distributed continual planning* in Section 1.4). To denote a particular set of the planning operations an agent realises, it is often convenient to define *agent roles*, presented in the next section.

3.5 Agent Roles in PrescriptiveCPS

Based on the supported set of planning operations, we define the number of *agent roles* to be adopted by the agent of PRESCRIPTIVECPS. These are summarized in Figure 3.6(a-d). An agent $a_x \in S$ might take one or more of these roles.

An agent in the **builder** role supports the *build* operation (see Figure 3.6a). As mentioned in Section 3.3, a flexibility model instance f for physical the system $P(f)$ can be built by (1) directly sensing $P(f)$ with electro-mechanical sensors, (2) using built-in knowledge about $P(f)$, (3) applying machine learning or predictive techniques such as *forecasting*, or (4) using the combination of these techniques. In fact, as we show later, the direct physical sensing and forecasting capabilities are important and must be distinguished. Therefore, we explicitly define two derivatives of the **builder** role:

- An agent in the (derived) **sensor** role is a builder agent which senses the physical (sub-)system $P(a)$ with electro-mechanical sensors and builds a flexibility model instance f such that $P(f) = P(a_x)$ (see Figure 3.7a).
- An agent in the (derived) **forecasting** role is a builder agent which uses forecasting techniques to build a flexibility model instance f (see Figure 3.7b). These forecasting techniques require external information and historical $P(f)$ measurements to predict stimuli and (in-)variant responses (future states of $P(f)$). The agent obtains such data from other agents and/or external sources via the informational link (see Section 3.1).

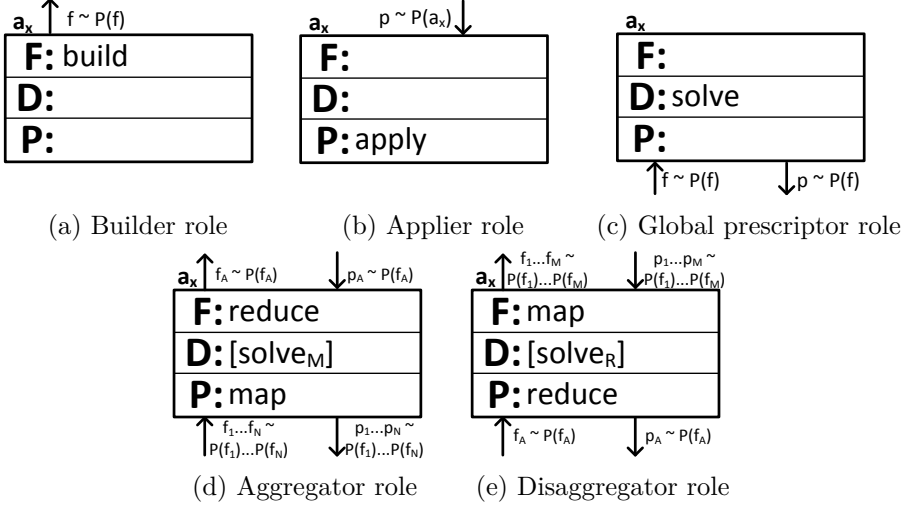


Figure 3.6: The core roles of agents in PRESCRIPTIVECPS

An agent in the **applier** role supports the *apply* operation (see Figure 3.6b). Consequently, it translates a prescription p into some actions (in some representation) leading a physical system $P(a_x)$ into the state represented by p . A special case of an applier agent is an *actuator agent* having a direct electro-mechanical link to make alternations in $P(a_x)$. This derived role is defined as follows:

- An agent in the (derived) **actuator** role is an applier agent having an electro-mechanical link to the physical (sub-)system $P(a_x)$ (see Figure 3.7c) and using this link to control $P(a_x)$ so that it follows a prescription p compatible with $P(a_x)$, i.e., $P(p) = P(a_x)$.

An agent in the **global prescriptor** role supports *solve* operation (see Figure 3.6c). It builds and solves decision model instances to produce prescription model instances according to flexibility model instances. The agent takes highest level decisions in PRESCRIPTIVECPS.

An agent in the **aggregator** role supports the *reduce*, (potentially) *solve_M*, and *map* operations with flexibility, decision, and prescription models, respectively (see Figure 3.6d). Consequently, it has the capability to aggregate flexibility model instances f_1, f_2, \dots, f_N into f_A and disaggregate a corresponding prescription model instance p_A into instances p_1, p_2, \dots, p_N such that $p_A, p_1, p_2, \dots, p_N$ match $f_A, f_1, f_2, \dots, f_N$, respectively.

An agent in the **disaggregator** role supports the *map*, (potentially) *solve_R*, and *reduce* operations with flexibility, decision, and prescription models, respectively (see Fig-

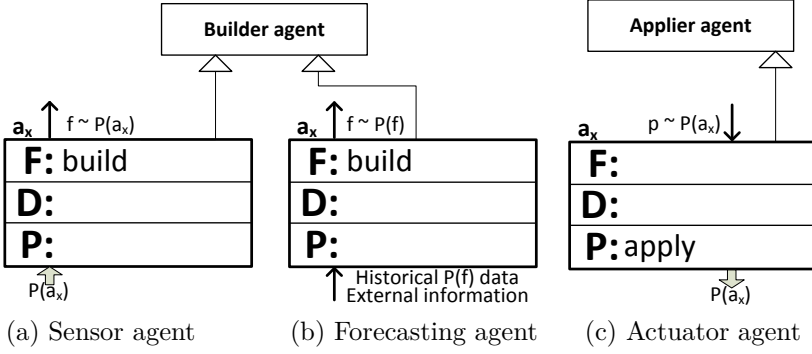


Figure 3.7: The special cases of the builder (*a* and *b*) and the applier (*c*) roles

ure 3.6e). Consequently, it has the capability to disaggregate a flexibility model instance f_A into instances f_1, f_2, \dots, f_M and aggregate corresponding prescription model instances p_1, p_2, \dots, p_M into an instance p_A such that $p_A, p_1, p_2, \dots, p_M$ match $f_A, f_1, f_2, \dots, f_M$, respectively.

Lastly, we use one more derived role, termed a **decision-making** role, to denote the *global prescription*, *aggregator*, and *disaggregator* roles as they take decisions to produce prescriptions. Note, this role name is also valid in those *aggregator* and *disaggregator* role cases when prescription models are built without the use of decision models (as $solve_M$ and $solve_R$ are not used). In these cases, there is only one feasible decision candidate, and it is deterministically chosen by *map* or *reduce*, as explained in Section 3.3.

3.6 MIRABEL ICT System as the Instance of PrescriptiveCPS

We now show that the MIRABEL ICT system (see Chapter 2) is the instance of PRESCRIPTIVECPS and that our proposed concepts can be conveniently used to describe such a complex system. First, we identify flexibility and prescription models managed by the agents of the MIRABEL ICT system. Then, we define the roles of these agents. Finally, we show the detailed model management workflow pursued by MIRABEL ICT system agents to balance electricity consumption and production.

Flexibility and prescription models in the MIRABEL ICT system

As explained in Sections 2.1–2.3, the MIRABEL ICT system manages different types of data concerning flexible and inflexible demand and supply loads, represented as *flex-offers/flex-offer assignments* and *time series*, respectively. All such data forms flexibility

Managed By	Visualisation of a physical domain	Model	Model Representation in MIRABEL
TSO agent		MBA flexibility model	<ul style="list-style-type: none"> • A time-series of Δ values (1) • Set of flex-offers (2)
		MBA prescription model	<ul style="list-style-type: none"> • Set of flex-assignments <p>aggregated loads prescribed to multiple units</p>
BRP agent		BG flexibility model	<ul style="list-style-type: none"> • Set of flex-offers (1) • Set of time-series (2)
		BG prescription model	<ul style="list-style-type: none"> • Set of flex-assignments <p>aggregated loads prescribed to multiple units</p>
Aggregator agent		Multi-unit flexibility model	<ul style="list-style-type: none"> • Set of flex-offers <p>flexible loads from multiple units</p>
		Multi-unit prescription model	<ul style="list-style-type: none"> • Set of flex-assignments <p>aggregated loads prescribed to multiple units</p>
Prosumer agent		Unit flexibility model	<ul style="list-style-type: none"> • A flex-Offe <p>flexible load from a single unit</p>
		Unit prescription model	<ul style="list-style-type: none"> • A flex-Offe assignment <p>load prescribed to a single unit</p>

Figure 3.8: Flexibility and prescription models managed by the MIRABEL's ICT system

and prescription models, shown in Figure 3.8, respectively. The number and the names of these models are specific to the MIRABEL use-case and are bound to the levels of the hierarchy of the MIRABEL ICT system. We now elaborate these flexibility and prescription models.

A **unit flexibility model** f^{unit} is represented by a flex-offer and it defines the flexible load of a single consumption and/or production unit such as a heat pump and electric vehicle (see Example 3.2). The corresponding **unit prescription model** p^{unit} is in the representation of a flex-offer assignment (see Example 3.4).

A **multi-unit flexibility model** f^{multi} is represented by the set of flex-offers, and it defines flexible loads collectively incurred by multiple consumption and/or production units (e.g., 200 heat pumps and 100 EVs). In the 5-tuple representation, $f^{multi} = (V, S, R, D^V, D^R)$ includes V and S defined as $V = \bigcup_{i=1..N} V_i$ and $S = \times_{i=1..N} S_i$, where $\forall i = 1..N$ ($V_i, S_i, R_i, D_i^V, D_i^R$) are the 5-tuple representations of flex-offers in the set. The response model R is defined as the sum of all individual responses R_i , $i = 1..N$, i.e., $R : (s_1, \dots, s_N) \mapsto R_1(s_1) \oplus \dots \oplus R_N(s_N)$, where $s_i \in S_i$, $\forall i = 1..N$, and \oplus is the operator for time series addition. The corresponding **multi-unit prescription model** p^{multi} is represented as the set of flex-offer assignments.

A **BG flexibility model** f^{BG} defines both flexible and inflexible consumption and production loads within a balance group (BG). It is represented by the set of flex-offers (for flexible loads) and the set of time series (for inflexible loads). Likewise in the multi-unit model, the response model of f^{BG} is defined as the sum of time series (using \oplus) and represents the total load of all underlying flexible and inflexible units within a BG. Like the *multi-unit prescription model*, the corresponding **BG prescription model** p^{BG} is

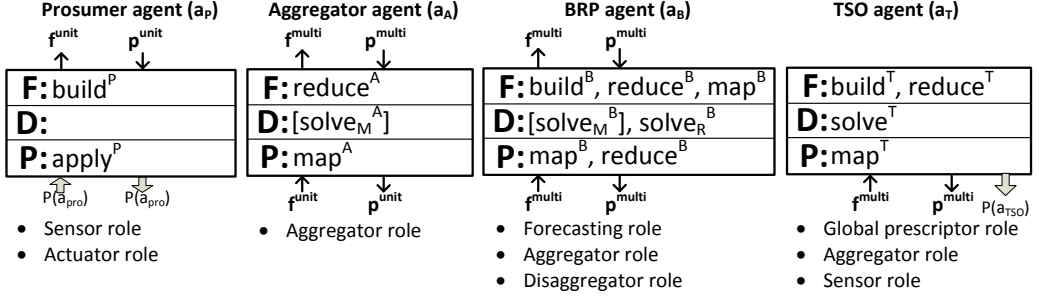


Figure 3.9: The roles of agents in the MIRABEL's ICT system

represented as the set of flex-offer assignments.

A **MBA flexibility model** f^{MBA} defines a difference between the consumption and production loads at the market balance area (MBA) level along with the flexible BG loads to be used for balancing. Consequently, f^{MBA} is represented as a single time series (for Δ loads) and the set of flex-offers (for flexible loads). In the 5-tuple representation, the model is defined similarly to the multi-unit and BG models. Depending on a chosen stimulus (in the format of flex-offer assignments), the response of f^{MBA} represents the consumption and production deviations at the MBA (grid) level, compensated with the loads of a BRP. Like the *multi-unit* and *BG* prescription models, the corresponding **MBA prescription model** p^{MBA} is represented as the set of flex-offer assignments.

MIRABEL agent roles and a model management workflow

The set of cyber-agents in the MIRABEL ICT system, denoted as S , forms a hierarchy such that the agent $a_x \in S$ has at most one immediate successor ($|master(a_x)| \leq 1$). The MIRABEL agents manage the described *unit*, *multi-unit*, *BG*, and *MBA* flexibility and prescription models using the planning operations, introduced in Section 3.3. The supported types of operations and the corresponding MIRABEL agent roles are shown in Figure 3.9. To balance consumption and production at the levels of a balance group and an MBA (power grid), the agents realize the workflow, shown in Figure 3.10. We now briefly overview the involved operations, grouped by the MIRABEL agent roles (which are orthogonal to the PRESCRIPTIVECPS roles).

A **prosumer agent** in the *sensor* and *actuator* roles realizes $build^P$ and $apply^P$ operations. It builds (see ① in Figure 3.10) a *unit flexibility model* (f_1^{unit}) for each prosumer's intent to use an appliance (e.g., EV) and applies (⑬) the corresponding *unit prescription model* (p^{unit}) to activate loads from the appliance, respectively. In the MIRABEL context, $build^P$ builds flex-offers and $apply^P$ applies flex-offer assignments, respectively.

An **aggregator agent** in the *aggregator* role realizes the $reduce^A$ and map^A (and

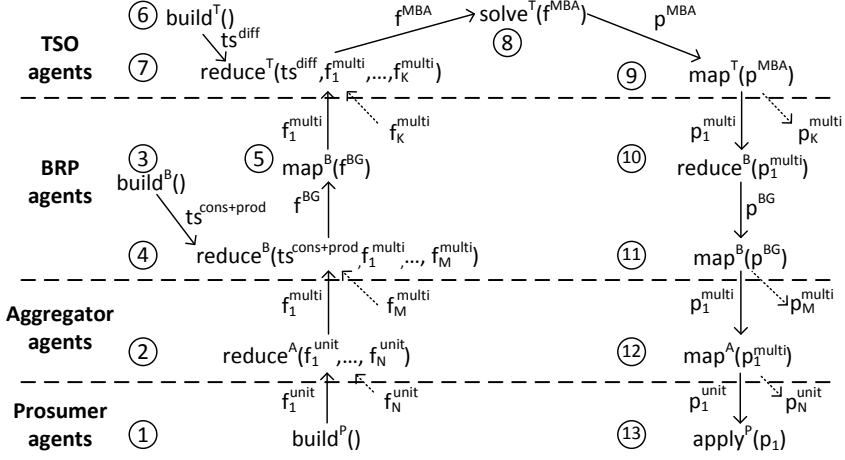


Figure 3.10: The workflow of flexibility and prescription model processing in MIRABEL's ICT system

optionally $solve_M^A$) operations. These aggregate (②) a set of *unit flexibility models* from different prosumer agents ($f_1^{unit}, \dots, f_N^{unit}$) into a single *multi-unit flexibility model* (f_1^{multi}) and disaggregates (⑫) a *multi-unit prescription model* p_1^{multi} into the set of *unit prescription models* ($p_1^{unit}, \dots, p_N^{unit}$), respectively. In the MIRABEL context, $reduce^A$ aggregates a set of flex-offers into another set of flex-offers and map^A disaggregates a set of flex-offer assignments into another set of flex-offer assignments (as explained in Section 2.2).

A **BRP agent** takes the *forecasting*, *aggregator*, and *disaggregator* roles. In the *forecasting* role, the agent uses $build^B$ to forecast (③) inflexible consumption and production loads in the format of a time series set ($ts^{cons+prod}$). In the *aggregator* role, the BRP agent uses $reduce^B$ to fuse (④) inflexible and flexible loads, represented as a set of time series ($ts^{cons+prod}$) and a set of multi-unit flexibility models from multiple aggregator agents ($f_1^{multi}, \dots, f_M^{multi}$), respectively. This action results into a *BG flexibility model* (f^{BG}). In the *disaggregator* role, the agent uses map^B to disaggregate (⑤) the *BG flexibility model* (f^{BG}) into a single *multi-unit flexibility model* (f_1^{multi}) representing flexible loads to be used at the MBA level (inter-balance group). Furthermore, with $reduce^B$ the agent aggregates (⑩) corresponding *multi-unit prescription models* into a *BG prescription model* (p^{BG}) so that both the agent's (BRP's) and the successor's (TSO's) balancing objectives are aligned. Finally, it uses map^B to disaggregate (⑪) a *BG prescription model* (p^{BG}) into the set of *multi-unit prescription models* ($p_1^{multi}, \dots, p_M^{multi}$). In the MIRABEL context, the agents perform scheduling of flex-offers while exchanging flex-offers and flex-offer assignments with aggregator and TSO agents, as explained in Section 2.2.

A **TSO agent** takes the *sensor*, *aggregator*, and *global prescriptor* roles. In the sen-

sor role, the agent uses $build^T$ to build ((6)) a flexibility model of real-time deviations between consumption and production at the MBA level (ts^{diff}). Then, in the *aggregator* role, it uses $reduce^T$ to fuse ((7)) the deviations model with the set of *multi-unit flexibility models* ($f_1^{multi}, \dots, f_K^{multi}$) representing flexible loads of different BRPs. This action results in an *MBA flexibility model* (f^{MBA}). Finally, in the *global prescriptor* role, the agent prescribes ((8)) the MBA flexibility model (f^{MBA}) to compensate imbalances at the grid level, and with map^T the agent disaggregates ((9)) the prescription model (p^{MBA}) into the set of *multi-unit prescription models* ($p_1^{multi}, \dots, p_K^{multi}$). In the MIRABEL context, the TSO thus uses flex-offers as alternative (secondary or ternary) reserve loads while balancing the grid.

As seen above, the MIRABEL ICT system consists of the set of agents in specific PRESCRIPTIVECPS roles and relies on the planning operations involving different flexibility and prescription (and decision) models. The MIRABEL project gives the rise to the set of high-level requirements for these agents, roles, and planning operations. In the next section, we generalize these requirements for other (similar) instances of PRESCRIPTIVECPS.

3.7 Common Requirements of PrescriptiveCPS

In this section, we draw high-level functional and non-functional requirements for PRESCRIPTIVECPS agents and agent roles. First, we define high-level requirements that are applicable to all agents of PRESCRIPTIVECPS and then we focus on individual agent roles (groups) and draw requirements specific to a particular role. We motivate and exemplify these requirements using the use-case of the MIRABEL project.

Common Agent Requirements

Requirement 1: *An agent of PRESCRIPTIVECPS has to provide near real-time model management and exchange capabilities.*

Example 3.11 (Performance requirement in MIRABEL). *As seen in the MIRABEL use-case, flexibility and prescription models have to be built, propagate through the hierarchy, and be applied fast enough so that decisions are realized using “fresh” rather than outdated models. This performance requirement applies to all agent roles, unexceptionally. If the requirement is not met, the MIRABEL ICT system, or any other instance of PRESCRIPTIVECPS, is not responsive to, for example, shortfalls in a physical power grid.*

Requirement 2: *An agent of PRESCRIPTIVECPS has to be sufficiently consistent with other agents (successors and predecessors) and/or a physical system.*

This is a correctness requirement ensuring that flexibility models (the view of a physical system) and prescription models (the view of the actuation of a physical system) are sufficiently consistent with a physical system and between agents. If sufficient consistency is achieved, decisions are made (and realized) based on models that correctly represent an underlying physical system and its actuations.

Example 3.12 (Correctness requirement in MIRABEL contra-example).

As a contra-example, consider a flexibility model (at the BRP level) with incorrect stimuli and/or responses of a balance group. This will result into imbalances in a physical balance group even if a decision model leads into a perfect balance.

Requirement 3: *An agent of PRESCRIPTIVECPS must be highly available and be able to tolerate network splits (especially at the higher levels).*

An instance of PRESCRIPTIVECPS is required to be reliable, especially at the higher levels. It is very important that the system provides the service to stakeholders from the higher hierarchical levels (e.g., BRPs, TSOs) with high availability. If some agent (e.g., a prosumer) disconnects from the system, both the agent and the rest of the system must be able to operate autonomously. The system must be resilient to failures such as network splits (partitioning) and agent crashes. Note, Requirements 2–3 are conflicting, as stated by the CAP theorem [33].

Requirement 4: *An agent of PRESCRIPTIVECPS must have (flexibility and prescription) model management and storage capabilities.*

Example 3.13 (Model management and storage in MIRABEL). *Consider the map^B and reduce^B operations for BG prescription models. They require an instance of the BG flexibility model to be kept (at least) until a respective instance of the BG prescription model is generated (as defined in Section 3.3). Additionally, as exemplified in Figure 3.11, multiple flexibility and prescription model instances may co-exist due to (1) changes in a physical system (e.g., new flex-offers) and (2) multiple decisions that may be generated with a single instance of a flexibility model (e.g., the BG flexibility model. As we show later, the history of model instances must be kept to be able to study and document past conditions that lead into corresponding decisions. An agent of the MIRABEL ICT system must be able to store, retrieve, and process multiple instances of (several) flexibility and prescription models (e.g., unit, multi-unit).*

Requirements for the Aggregator Role

As defined in Section 3.5, an aggregator agent implements the pair of *reduce* and *map* operations with flexibility and prescription models, respectively. The pair has to be

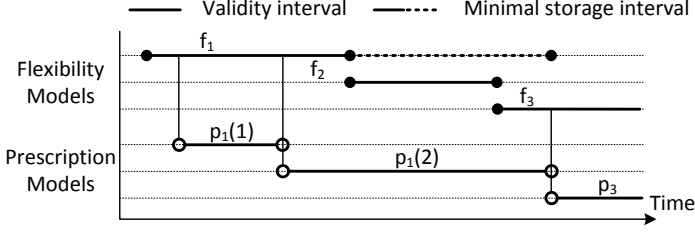


Figure 3.11: The validity and storage times of flexibility and prescription model instances

(carefully) designed so that the governing correctness requirement (Requirement 2) is satisfied, and, consequently, the views of a physical system before and after *reduce* and the views of a physical system actuation before and after *map* correspond. However, there are additional requirements related to the practical use of *reduce* and *map* (and the whole PRESCRIPTIVECPS).

When the set of flexibility models are aggregated with *reduce*, this might lead into a (compound) flexibility model, which is difficult or infeasible to handle at (higher) decision levels due to its size or a (very) high number of prescription (unknown/decision) variables and the (exploded) number of alternative stimuli. It is, however, often preferable to keep the number of the prescription variables at the manageable level while still preserving (lots of) alternative physical system responses, as they need to be explored in decision making. We formulate this aim using the following two conflicting requirements:

Requirement 5: *An aggregation agent must reduce the number of stimulus alternatives (prescription variables) during the reduce of flexibility models.*

Requirement 6: *An aggregation agent has to preserve as many response alternatives as possible during the reduce of flexibility models.*

Example 3.14 (Aggregator role requirements in MIRABEL). *Consider the MIRABEL use-case with millions of prosumers at the lowest level. It is, however, very impractical to deal with the very large number of flex-offers at the BRP or TSO level. This is why aggregator agents of the MIRABEL ICT system use lossy reduce^A to aggregate unit flexibility models into multi-unit flexibility models, while building “larger flex-offers”, as shown in Figure 2.6. The lossless reducers (reduce^B and reduce^T) are still practically feasible and used, e.g., to fuse flexibility models of different types (e.g., $ts^{\text{cons+prod}}$ and f_1^{multi} to f^{BG} or ts^{diff} and f_1^{multi} to f^{GRID}).*

Requirements for the Forecasting Role

As defined in Section 3.5, a forecasting agent implements the *build* operation, which builds a flexibility model by projecting historical data to future and thus predicting expected responses (and stimuli) of a physical (sub-)system. This leads to the following requirement:

Requirement 7: *A forecasting agent must support a forecasting functionality (e.g., time series forecasting), which allow building responses (and stimuli) of a flexibility model based on historical measurements (of a response/stimulus), external data, and relevant context information.*

As most forecasting techniques require a long series of data to produce accurate forecasts [34], the forecasting agent has to satisfy an additional storage requirement.

Requirement 8: *A forecasting agent must have sufficient storage for historical measurements, external data, and all required contextual information.*

Example 3.15 (Forecasting role requirements in MIRABEL). *As seen in Section 3.6, the forecasting operation ($build^B$) is used to build a “missing” flexibility model ($ts^{cons+prod}$) for a physical system (non-flexible demand and supply) that is not directly observed by underlying predecessor agents but is required to “complete” a larger (f^{BG}) flexibility model. Such capability to build “missing” models is required in MIRABEL ICT system as well as in other instances of PRESCRIPTIVECPS.*

Requirements for the Decision-Making Role

As described in Section 3.5, the *decision-making* role covers the *global prescription*, *aggregator*, and *disaggregator* roles, and a corresponding decision-making agent uses *solve*, *solve_M*, or *solve_R* (with exceptions) to produce prescriptions, respectively. As explained in Section 3.3, the *solve*, *solve_M*, and *solve_R* operations produce prescriptions in two steps: (1) *decision model building* (2) *decision model solving* (evaluation). The *decision model building* can be seen as the application of a “builder function” ($build(f, \dots)$) which produces an instance of a decision model according to a flexibility model instance f and is carefully designed by humans to reflect their desired (external) objectives and/or constraints, e.g., minimize imbalances between consumption and production. To design such builder function (*build*), a decision-making agent should provide to (human) users sufficient insights into a decision domain, allowing them to design effective decision models (to be generated with *build*). Such insights may include, for example, the graphical visualization of decision spaces (flexibility models). They may allow designing new and/or updating existing decision model builder functions. The need for the described analytical capabilities is formulated as the following requirement:

Requirement 9: *A decision-making (global prescription, aggregation, or disaggregator) agent must offer a decision model building and adaptation support (analytics) to*

(human) users.

The decision model solving can be done utilizing the capabilities of an optimization problem solving. Consequently, the agent is required for the built-in support of such capabilities, as formulated by the following requirement:

Requirement 10: *A decision-making agent (a global prescription, aggregation, disaggregator agents) must have the capability of solving decision (optimization) model instances.*

Example 3.16 (Decision-making role requirements in MIRABEL). *A BRP agent's decision model for minimizing imbalances between consumption and production was carefully designed in the MIRABEL project, and three meta-heuristic scheduling algorithms (a randomized greedy search, an evolutionary, and a hybrid algorithm) were proposed [35] to solve instances of this decision model.*

3.8 Summary

In this chapter, we have presented PRESCRIPTIVECPS, which is our proposed (conceptual model of a) CPS for distributed continual planning and control. In PRESCRIPTIVECPS, agents are organized hierarchy based on their obedience to each other. They manage a number of physical system model instances. We have defined and elaborated the three types of these models, namely *flexibility*, *decision*, or *prescription* models. To deal with these types of models, an agent uses the set of planning operation. Depending on the set of supported operations, the agent takes one of more of *roles* such as the *builder* (*sensor*, *forecasting*), *applier* (actuator), *global prescription*, *aggregator*, or *disaggregator* role. We also demonstrated that the MIRABEL ICT system is a specialized instance of PRESCRIPTIVECPS. By using the MIRABEL use-case example, we formulated 10 general requirements that are applicable to similar instances of PRESCRIPTIVECPS and their agents. In the next chapter, we focus on the software of a PRESCRIPTIVECPS agent, while aiming to support these requirements.

Chapter 4

Agent Software System for Prescriptive Analytics

We now focus on the software of a single PRESCRIPTIVECPS agent and present the generic (internal) architecture of an agent software system. The architecture supports multiple agent roles and, as we elaborate later in the thesis, allows meeting the most of the requirements specified in Section 3.7. In this chapter, first we present the architecture focusing on the logical separations of data management functions and then show how various components of the architecture map to the agent model (defined in Section 3.1). Then, we overview the capabilities of a database management system (DBMS) component, enhanced to support specific operational and analytical tasks of a PRESCRIPTIVECPS agent. We overview the analytical capabilities and the types of queries supported by the DBMS. Finally, we point-out agent (sub-)components, which we focus in details in the rest of the thesis.

4.1 Architecture of an Agent Software System

We design software system architecture for a PRESCRIPTIVECPS agent based on the standard three-layer (three-tier) model [36]. The architecture is shown in Figure 4.1. It is designed to be generic to handle all the agent roles described in Section 3.5 and to address the role-independent requirement for model storage (Requirement 4) and the role-specific requirements for forecasting (Requirement 7), historical data storage (Requirement 8), and decision model building and solving (Requirements 9–10).

The architecture includes the standard three *data management*, *business logic*, and *presentation* layers with the number of components designed or customized specifically for the agent of PRESCRIPTIVECPS. We now briefly overview these components in each

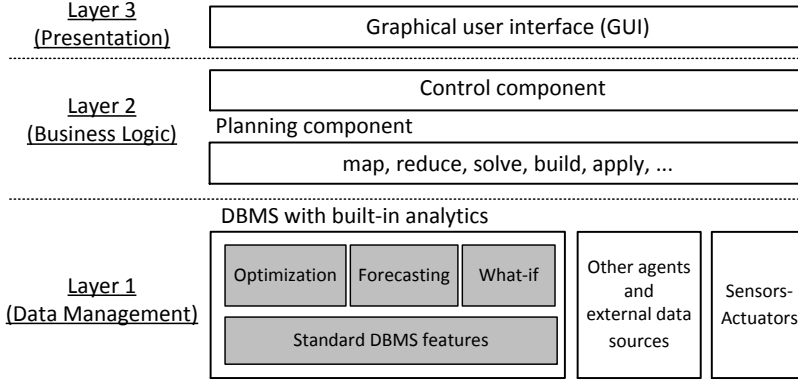


Figure 4.1: The architecture of a PRESCRIPTIVECPS agent software system

layer, while elaborating them in-depth in the current and the remaining chapters of this thesis.

Data Management Layer includes data sources and data destinations of an agent. These are *sensors/actuators*, *other agents*, and an integrated general-purpose database management system (DBMS) with built-in analytical capabilities. The DBMS, in addition to the standard features, supports forecasting (*Forecasting*), optimization problem solving (*Optimization*), and *what-if* analysis based on so-called *hypothetical scenarios* [37] (*What-if*).

Business Logic Layer includes components realizing role- or agent-specific business objectives and rules (e.g., of a TSO or BRP actor). Specifically, the layer includes the *control component* and the *planning component*. Here, the control component orchestrates the data exchange between different data sources and destinations. The planning component, among other functionality, offers a concrete realisation of the (sub-)set of the planning operations (see Section 3.3) such as *map*, *reduce*, *solve*, *apply*, and *build* (of the sensor/forecasting roles).

Presentation Layer includes a graphical user interface (GUI) allowing (human) users to query and visually analyse data of an agent.

Figure 4.2 shows how various components of the software architecture can be integrated into an agent, defined in Section 3.1. Depending on the set of roles an agent takes, the agent may or may not integrate all of the described (sub-)components. For example, an agent integrates the control and the DBMS components irrespective to its role (see *mandatory components*). However, the most of these components are not integrated in a (simple) agent taking the *sensor* or *actuator* roles and requiring no analytical capabilities. In the latter case, the agent may only integrate sensors, actuators, the control component, and a standard DBMS with no analytical capabilities.

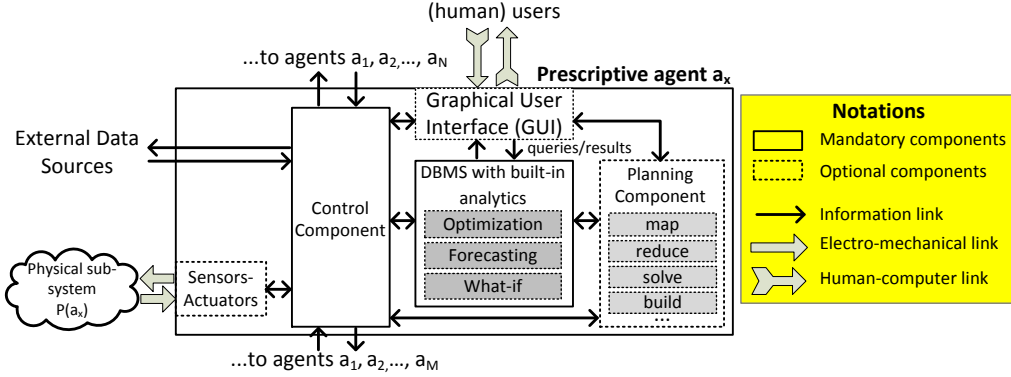


Figure 4.2: The integration of the agent software system with other PRESCRIPTIVECPS agent components

We now focus exclusively on the DBMS component with the built-in *forecasting*, *optimization*, and *what-if* capabilities. While focusing on the component, we elaborate the supported types of analytical queries, potentially, contributing the decision model building and adaptation (Requirement 9).

4.2 Analytical Capabilities of the DBMS

The DBMS of a PRESCRIPTIVECPS agent software, among other types of data, stores historical measurements and the flexibility and prescription models of different types (Requirement 4 and Requirement 8). The supported analytical capabilities (*forecasting*, *optimization*, and *what-if*) enable novel and powerful ways to analyse all such data, thus helping (human) analytics/administrators (1) to study the (past, current, and future) performance of a physical system and (2) to optimize the PRESCRIPTIVECPS agent operation through more effective decision models (Requirement 9). We now focus on each of these capabilities individually while briefly presenting functionality and advantages they bring.

The **forecasting** capability allows processing a large amount of historical data inside a DBMS to produce accurate forecasts without the need to export data to external forecasting tools (e.g., Matlab). As elaborated in Chapter 7, this functionality offers increased usability, productivity, and performance (Requirement 1), and can be effectively used for realizing the *build* operation (for forecasting) in the planning component.

The **optimization** problem solving functionality enables complex analytical queries involving optimization problem solving to be processes inside a DBMS without the need to export data to external optimization tools (e.g., AMPL). As will be elaborated in Chapter 8, this brings many advantages such as increased (DBMS and solver) usability.

ity, (user) productivity, and (overall) performance (Requirement 1). This functionality can be effectively used to realise the *solve*, *solve_R*, and *solve_R* operations in the planning component, for building and solving decision models based on flexibility models (and other inputs).

The **what-if** analysis allows assessing the effects of various hypothetical scenarios, which, as we elaborate in Chapter 9, are user-defined as ordered sets of hypothetical modifications (deletions, insertions, and updates) of data in a database. The convenience of such hypothetical scenarios in the process of data analysis is shown in the following example:

Example 4.1 (What-if analysis in the MIRABEL use-case). *Consider a (human) analyst in the MIRABEL use-case willing to study consumption and production imbalances in a balance-group (BG) as if no production from RES was produced at a particular day in the past. In such case, the analyst may choose (or define) and apply a hypothetical “no-wind” scenario and then issue a query(-ies), retrieving the amount(-s) of imbalances encountered as if such scenario really happened.*

The *time-travel* functionality [38] (**FOR SYSTEM_TIME AS OF, FOR BUSINESS_TIME AS OF**) of existing bi-temporal databases can be seen as the special case of scenario-driven querying, during which the database is queried as if system or business time is as user specifies in its scenario. Such a “time-travelling” well complements what-if analysis in PRESCRIPTIVECPS, as demonstrated in Chapter 9.

In the next section, we overview the most important types of queries supported by the DBMS utilizing the described analytical capabilities.

4.3 Queries Supported by the DBMS

We now outline the most important types of queries to be supported by the DBMS of a PRESCRIPTIVECPS agent. We list these query types in the order of their complexity, and include the types that can be supported by standard DBMSs (without analytical capabilities) as well as by the DBMS with the described analytical capabilities (*optimization, forecasting, and what-if*).

History queries are standard DBMS queries focusing on the (factual) data stored in a database. Among other purposes, historical queries may be used to study “deviations from a plan” while assessing the (past) performance of PRESCRIPTIVECPS.

Example 4.2 (A history query). *As explained in Section 3.1, an agent collects flexibility model instances, which (in the invariant part) may represent (historical) measurements of a physical system state. A historical query may contrast such measurements against (existing) prescription model instances that, potentially, led the physical system into the states represented by these measurements.*

Forecast queries project physical system responses (and stimuli) into future based on historical measurements (and other context information) in a database. Such queries rely on either built-in or external *forecasting* techniques and may be used to study expected behaviour of a physical system.

Example 4.3 (A forecast query). *A forecast query may retrieve (expected) day-ahead values of an inflexible RES production ($ts^{cons+prod}$) based on historical measurements stored in a database.*

Stimulus queries provide overviews of stimuli defined by flexibility model instances stored in a database. Such queries can, potentially, be used in feasibility/risk analysis where the flexibility of a physical system is explored.

Example 4.4 (A stimulus query). *A stimulus query may retrieve the sums or the distributions of time and amount flexibilities (as defined in Section 2.1) for a given set of flex-offers (a multi-unit model) as the estimate of the total flexibility (capability to change state) of a physical system.*

Response queries provide overviews of responses defined by flexibility model instances stored in a database. Such type of queries can, potentially, be used in feasibility/risk analysis where alternative (incl., nominal or peak) responses of a physical system are explored.

Example 4.5 (A response query). *A response query may retrieve a time series (response) with feasible minimum, maximum, or average energy amount values at every time interval, allowed by any feasible stimuli in a given set of flex-offers (a multi-unit model).*

Prescription queries allow studying (potential) prescriptions, derived “on-the-fly” during the query evaluation. Such queries may rely on the built-in support for *optimization* and can be used to study impacts of various decision models. The *solve*, *solve_R*, and *solve_M* operations of the planning component (see Section 4.1) are based on such queries.

Example 4.6 (A prescription query). *Given the set of flex-offers (a multi-unit model) stored in a database and a user-specified objective for balancing (Chapter 8 describes how such an objective is specified), a prescription query may retrieve the amount of remaining imbalances incurred by the best found set of flex-offer assignments (a stimuli/prescription).*

Prescription adaptation queries allow studying feasible changes (deltas) of the prescription model that is already generated and stored in a database together with a corresponding flexibility model. Such queries may also rely on the built-in *optimization* functionality.

Example 4.7 (A prescription adaptation query). *A prescription adaptation query might retrieve energy amount values that can still be increased or decreased at a given time interval with the respect to existing prescription, e.g., to counter unexpected highs or lows in the RES production.*

What-if queries allow studying flexibility, decision, and prescription models (incl., measurements) as if some user specified or selected hypothetical scenario has, in fact, really happened, as presented by Example 4.1. Among these scenarios, there exist scenarios where a business or system time is (hypothetically) set to a user-specified time moment. Such “time-travel” scenarios allow retrieving measurements, flexibility, or prescription models in versions that were valid or present in a database at that particular time moment. Such queries rely on the built-in support for *scenarios* and can be combined with other types of queries such as *forecast queries* and *prescription queries*. When combined with *forecast* and *prescription queries*, hypothetical forecasts (flexibility models) and subsequent prescriptions can be studied, for example, as if measurements followed a different user-specified pattern or alternative decision models (decision model builders of *solve*) were applied in the past or will be applied in future.

Example 4.8 (A what-if forecast and prescription query). *A combined what-if, forecast, and prescription query can be used to study hypothetical imbalances as if a past day’s electricity consumption followed a different pattern (than it actually did) leading to different forecasts, different decisions, and different imbalances for the day-ahead.*

In PRESCRIPTIVECPS, the DBMS of an agent in the *decision-making* role supports all the discussed types of queries and allows mixing standard DBMS queries with the queries described above. When an agent is equipped with such a DBMS, an analyst (human) is provided with the rich set of *prescriptive analytics* tools for both managing and analysing flexibility, decision, and prescription model instances in a single powerful data management and analysis system. Furthermore, pushing more of the analytical functionality into a single common system allows for more simplified and more lightweight implementations of the *GUI*, the *control*, and the *planning* components (see Figure 4.1), as the required analytical functionality can be reused and shared across all these components. Therefore, the (large) part of this thesis (Chapters 7–9) is devoted for supporting all the presented types of queries in a single integrated DBMS for planning. Thesis contributions on (the rest of) agent’s software components are summarized in the next section.

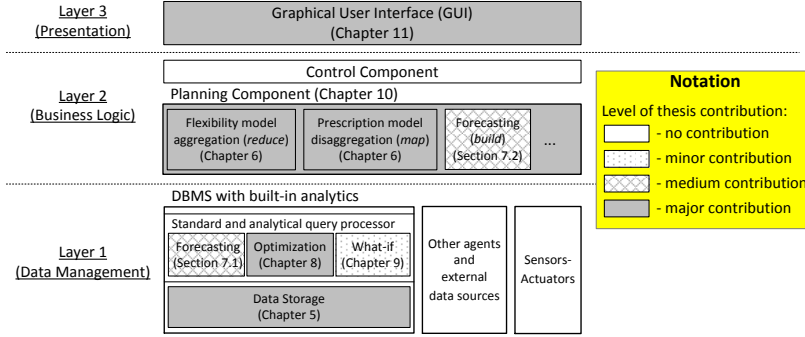


Figure 4.3: Thesis contributions on different components of the PRESCRIPTIVECPS agent software

4.4 Contribution on the Agent Components

As shown earlier, PRESCRIPTIVECPS agent software consists of the number of components (and functionalities) at all three layers of the agent software architecture. The components (and functionalities) from the *data management* layer are *generic*, meaning that they can, potentially, be reused across all agents (and agent roles) of PRESCRIPTIVECPS. On the contrary, the components (and functionalities) from the *business logic* and *presentation* layers are *specific* to an agent (or an agent role) and, therefore, must be specially adapted and customised for a particular agent (or an agent role).

The rest of the thesis focuses on, and provides contributions for, both the *generic* and *specific* components (and their functionalities) at all the three layers, as shown in Figure 4.3. Specifically, Section 7.1 and Chapters 8–9 consider the generic DBMS and focus on its built-in *forecasting*, *optimization*, and *what-if* functionalities. Chapter 5 shows how this generic DBMS component can be used to store agent-specific data using a generalised database schema that can, potentially, be reused across all agents. Then, Chapter 6 presents agent-specific techniques to aggregate flexibility model instances (using *reduce*) and disaggregate prescription model instances (using *map*). Similarly, Section 7.2 presents agent-specific (domain-specific) flexibility model forecasting techniques, which rely on the forecasting-capable DBMS (Section 7.1). Chapter 10 considers the general architecture of the agent-specific planning component, and, finally, Chapter 11 elaborates agent-specific flexibility and prescription model visualisation and visual analysis techniques.

4.5 Summary

In this chapter, we presented the three-layer software architecture of PRESCRIPTIVECPS agent software and showed how various components of the architecture can be integrated into a PRESCRIPTIVECPS agent. Then, we focused on the DBMS component and overviewed its supported types of queries and PRESCRIPTIVECPS-specific functionalities that have to be integrated into the DBMS for convenient and efficient management and analysis of flexibility, decision, and prescription models. Finally, we presented generic and agent-specific software components (and functionalities), which we focus on in the remaining chapters of this thesis in details.

Chapter 5

Multi-Dimensional Flexibility and Prescription Model Representation

In this chapter, we focus on the DBMS component of a PRESCRIPTIVECPS agent software system (see Section 4.1) and show how flexibility and prescription model instances and historical measurements can be stored in a database underlying this component (Requirement 4 and Requirement 8). For specificity, we focus on flexibility and prescription models from the MIRABEL use-case. There, instances of *unit*, *multi-unit*, *BG*, and *MBA* flexibility and prescription models together with other related data need to be stored by agents taking the roles that are defined by the Harmonized Electricity Market Role Model [27]. Flexibility and prescription models in MIRABEL are complex multi-dimensional entities represented as (sets of) flex-offers, (sets of) flex-offer assignments, and (sets of) time series (Section 3.6). For the storage of these, a multi-dimensional representation is considered, and the schema of a multi-dimensional warehouse (DW) is proposed. In this chapter, we first discuss the modelling of actors and their roles (Section 5.2) as well as flex-offers and time series (Sections 5.3–5.4). Then, we show the complete schema (Section 5.5) and experimentally evaluate alternative data modelling strategies based on typical queries from the MIRABEL use-case (Section 5.6). Finally, we review related work on energy-related data modelling (Section 5.7) and discuss flexibility and prescription model storage in other similar instances of PRESCRIPTIVECPS (Section 5.8). The content of this chapter is based on Publication [2].

5.1 Data Storage in the MIRABEL Use-case

As explained in Chapter 4, the DBMS component of a PRESCRIPTIVECPS agent software system handles the storage of data required for (online) planning and (offline) human-based analysis. To provide the storage of flexibility models, prescription models (Requirement 4), and measurements (Requirement 8) a *database* is required. In the general case, the schema of such database is specific to an agent (or a role) and depends on a number and types of flexibility and prescription models an agent needs to manage. In the MIRABEL use-case, fortunately, the (*unit*, *multi-unit*, *BG*, and *GRID*) flexibility and prescription models are made from (represented by) common and shared entities (“building blocks”) which are flex-offers, flex-offer assignments, and time series (see Section 3.6). Consequently, the common database schema suitable for all agents in any role can be designed in the MIRABEL use-case. We now elaborate the schema of such unified MIRABEL database. To design this schema, we rely on the Harmonized Electricity Market Role Model [27] and the MIRABEL specification [39].

In the MIRABEL use-case, an agent manages a data of a single actor (legal entity) in a particular role such as the *prosumer*, *aggregator*, *BRP*, or *TSO* role (see Section 2.2). When an agent communicates data with other agents, it deals with data for/from other *actors*, which can, potentially, take multiple *roles*. To be able to store local data and to keep track of exchanged data, the schema of the agent’s database must be designed to support multiple *actors* in multiple *roles* so that data is stored in a database for each actor-role combination separately. As the database integrates data from multiple sources (actor-roles), it can be seen as an agent’s (local) *data warehouse* (DW), and, therefore, referred as MIRABEL DW.

Example 5.1 (The MIRABEL DW of an aggregator agent). *As explained in Section 2.1, the agent of an aggregator actor exchanges data with the agents of BRP actors (successors) and the agents of prosumer actors (predecessors). The agent needs a DW for the storage of data of its own (aggregator) role as well data (flexibility and prescription models) interchanged between each successor or predecessor actors (agents) in the BRP and prosumer roles. Consequently, data in the MIRABEL DW need to be stored in “per actor in a role” fashion.*

We present how actors and actor roles are modelled in the MIRABEL DW, in Section 5.2. For each actor-role, MIRABEL DW needs to store instances of the flexibility and prescription models of the types *unit*, *multi-unit*, *BG*, and *MBA*, composed of (sets of) flex-offers, (sets of) flex-offer assignments, and (sets of) time series. As these are complex multi-dimensional entities, we elaborate the multi-dimensional representation of flex-offers and flex-offer assignments in Section 5.3, and the multi-dimensional representation of time series in Section 5.4. Finally, we present the full multi-dimensional MIRABEL DW in Section 5.5.

5.2 Modelling of Different Actors and Roles

The MIRABEL DW needs to support different actors taking different roles. To model these, we rely on (and aim to be consistent with) the Harmonized Electricity Market Role Model [27], which is generic, elaborate, and not limited only to the roles (*prosumer*, *aggregator*, *BRP*, *TSO*) presented earlier in Section 2.1. The role model defines additional roles (e.g., *consumer*, *producer*, *imbalance settlement responsible*, and *market operator*) and specifies more complex relationships and interaction patterns between all these roles. In our modelling, we aim to capture all relevant roles and show how these roles are inter-related and linked with *market balance areas* (MBAs) and *balance groups* (BGs), described earlier in Section 2.1. To achieve the consistency with both the role model and the MIRABEL specification [39], we use interchangeably the terms (1) *actor* and *legal entity*, (2) *actor role* and *legal entity role*, (3) *prosumer* and *party connected to grid*, (4) *transmission system operator* (TSO) and *system operator*, and (5) *aggregator* and *metered data aggregator*.

In the MIRABEL DW, we represent actors and the roles from the Harmonized Electricity Market Role Model by means of the tables, shown in Figure 5.1.

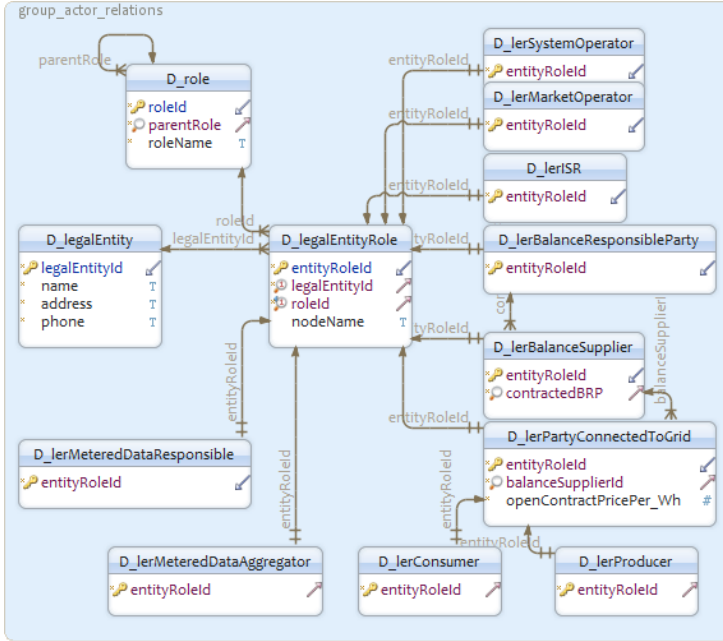


Figure 5.1: Tables for representing different actors/roles of the European electricity market

The table **D_role** represents roles such as *producer*, *consumer*, and *BRP*. A role can

belong to another parent role, and this is captured by a self-reference.

Example 5.2 (Consumers and producers as parties connected to a grid).

The consumer and producer roles are specializations of the role party connected to grid (prosumer). This is indicated by a self-reference in D_role .

Actors are represented by $D_legalEntity$. To capture when a certain actor plays a certain role, we use $D_legalEntityRole$. This table references both D_role and $D_legalEntity$. Further, it has an attribute to hold a unique ID for a given actor playing a given role. We include this ID as it makes it easy to point to an actor in a certain role. We do exactly that from a number of tables, as shown in Figure 5.1. For each role, there is a specialized table (D_*) that (directly or indirectly through another table) references $D_legalEntityRole$. A particular specialized table can be referenced, thus, making it explicit what kind of role is required.

Example 5.3 (Referencing for specialization). *The tables $D_lerProducer$ and $D_lerConsumer$ reference $D_lerPartyConnectedToGrid$ to indicate that actors in the producer and consumer roles are special cases of actors in the party connected to grid (prosumer) role.*

Example 5.4 (Referencing for an actor in a particular role). *The table $D_lerBalanceSupplier$ references $D_lerBalanceResponsibleParty$ to link actors in the balance supplier role with actors in the balance responsible party (BRP) role.*

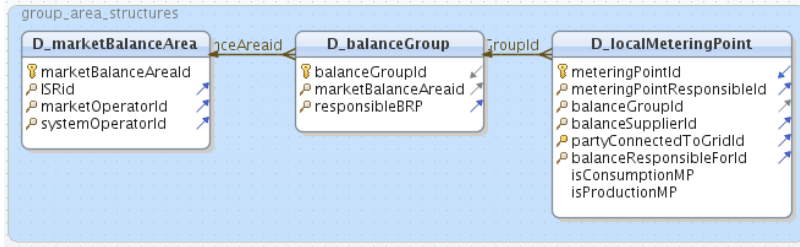


Figure 5.2: Tables for representing market areas

We provide tables to represent *market balance areas* (MBAs), as shown in Figure 5.2. Here, $D_localMeteringPoint$ represents the meters that are connected to the grid both at the producer and consumer sites. $D_localMeteringPoint$ references four different specializations of $D_legalEntityRole$. Further, it references $D_balanceGroup$ which in turn references $D_marketBalanceArea$ that hierarchically groups metering points, as explained previously in Section 2.1.

Discussion

The MIRABEL DW captures (1) actors in different hierarchically organized roles, (2) actor relationships at the role level, and (3) actor-role belonging to various domains within the power grid (MBA or BG). The schema allows representing attributes that are only relevant for certain roles. This is achieved by introducing specialized tables and then referencing these tables with foreign keys to indicate what kind of role is required. The schema thus helps to avoid mistakes where, e.g., an actor in the *balance supplier* role is referenced where an actor in the *balance responsible party* role actually should have been referenced instead. We note that if no special attributes must be stored for different roles, then, instead of storing the D_ler*'s as physical tables, they can be views selecting from D_legalEntityRole. This reduces the risk of mistakes further and makes maintenance of them automatic.

5.3 Modelling of Flex-offers and Flex-offer assignments

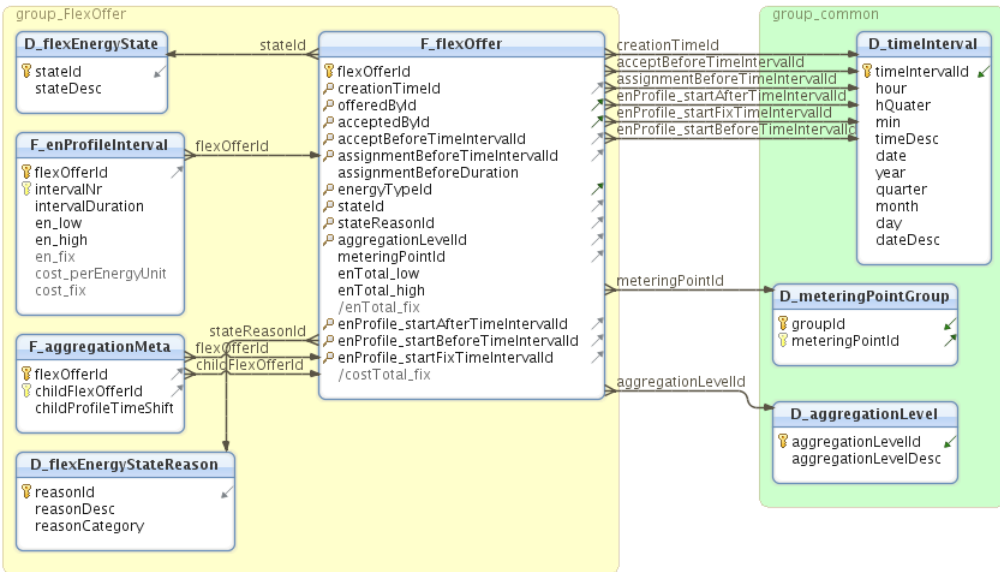


Figure 5.3: Tables for representing flex-offers

The MIRABEL DW needs to store flex-offers and flex-offer assignments that are core building blocks of the *unit*, *multi-unit*, *BG*, and *MBA* flexibility and prescription models

in the MIRABEL use-case. In addition to flex-offer (and flex-offer assignment) attributes described in Section 2.1, the MIRABEL specification [39] describes additional flex-offer (and flex-offer assignment) attributes (e.g., states, aggregation level) which we include in our modelling. To represent flex-offers and flex-offer assignments (both aggregated and non-aggregated), we use a multi-dimensional schema, shown in Figure 5.3. We first describe the dimension tables (which are recognized by the prefix `D_` in their names) and then the fact tables (recognized by the prefix `F_` in their names).

Dimension tables

As seen in Figure 5.3, all dimension tables (prefixed with `D_`) have surrogate keys with names ending with `Id`. The possible states for a flex-offer (e.g., *offered*, *accepted*, *assignment*, and *rejected*) are represented in the dimension `D_flexEnergyState` and all possible reasons that led the flex-offer to a particular state are represented in the dimension `D_flexEnergyStateReason`. The `D_flexEnergyStateReason` includes columns for both the generic categories and the specific reasons. This allows specifying the hierarchy of reasons with few generic reason categories and many more specific reason descriptions.

Example 5.5 (A flex-offer state and a state reason). *A flex-offer may be rejected if the requested energy is too high. In such case, the flex-offer has the state rejected, “Energy too high” as a generic reason, and “Energy (100kWh) too high” as a specific reason.*

We represent time by discretized time intervals. This is done by `D_timeInterval` which represents 15 minutes intervals (other interval lengths can be chosen if needed). Flex-offers are always related to at least one metering point (at the location where the energy is to be consumed or produced), but if a flex-offer is aggregated, it will be associated with many metering points. To capture this, `D_meteringPointGroup` is used as bridge table [40] between the fact table and `D_meteringPoint` which represents the individual metering points. To represent the aggregation level of a flex-offer, `D_aggregationLevel` is used.

Fact tables

As flex-offers and flex-offer assignments are often handed in pairs, the fact table `F_flexOffer` holds combined flex-offer and flex-offer assignment facts. The fact table references all the previously described dimension tables. The table holds only the current information about the pairs of flex-offers and flex-offer assignments, but, as we elaborate later, may include the standard *validity time* and *transaction time* attributes of bi-temporal databases for versioning. We now present the attributes relevant to a flex-offer and then elaborate the attributes for a flex-offer assignment.

Flex-offer representation

There are six foreign keys to `D_timeInterval` to represent different times such as when the flex-offer was created and when it at the latest has to be assigned, etc. These foreign keys thus all represent an absolute time. There is also an attribute `assignmentBeforeDuration` which holds a time span telling how long before the actual execution time the flex-offer assignment must be generated.

Further, `F_flexOffer` references `D_legalEntityRole` (explained in Section 5.2) twice to represent who offered and accepted the flex-offer, respectively. There are measures to hold the lowest and highest amount of energy required by a flex-offer. Finally, each represented flex-offer is given a unique identifier in the attribute `flexOfferId` which technically is a degenerate dimension.

Information about the profile intervals (slices) of flex-offers is represented in the fact table `F_enProfileInterval`. This fact table only has a single foreign key which references the unique `flexOfferId` in `F_flexOffer`. The imported value together with a sequential `intervalNr` forms the primary key for `F_enProfileInterval`. The reason for this design is that a single flex-offer can have many profile intervals (slices). For each represented profile interval, there is a duration specifying how many time units the profile interval spans over, and both the lowest and highest amount of energy needed in this interval. An alternative to this design would be to represent the measures of `F_enProfileInterval` in *arrays* in `F_flexOffer` such that all data about a given flex-offer would be represented in a single fact. Yet another alternative would be to represent all attributes of `F_enProfileInterval` in `F_flexOffer`, i.e., denormalize the data and have one (wide) fact in `F_flexOffer` for each profile interval. These three design alternatives are experimentally evaluated later in Section 5.6.

As flex-offers can be aggregated into larger flex-offers, we also introduce the table `F_aggregationMeta` which references `F_flexOffer` twice to point to the aggregating “parent flex-offer” and the smaller “child flex-offer” which has been aggregated, respectively. Profiles of each child flex-offer can be shifted relatively to the profile start of the parent flex-offer when aggregating child flex-offers into the parent. Therefore, for every child flex-offer, the `childProfileTimeShift` attribute indicates the amount of time units the profiles of the child flex-offer has been shifted in the aggregated flex-offer. This information is used in the disaggregation.

Flex-offer assignment representation

In the discussed `F_flexOffer` and `F_enProfileInterval` tables, there are additional attributes that are specific to a flex-offer assignment. The attribute `enProfile_startFixTimeIntervalId` in `F_flexOffer` represents a “prescribed” *starting time* of an underlying appliance. Similarly, `en_fix` in `F_enProfileInterval` is a measure holding the “prescribed” *energy amount* for each interval (slice) of a profile. Finally, there are computed (derived) measures (`/enTotal_fix` and `/costTotal_fix`) for representing the “prescribed” total energy amount

and price (when `cost_perEnergyUnit` is specified).

Example 5.6 (Representation of flex-offer and flex-offer assignment).

For a single flex-offer and a corresponding flex-offer assignment, there is a single fact in the `F_flexOffer` table and multiple facts in the `F_enProfileInterval` table. As flex-offer assignment is generated according to a flex-offer, each `enProfile_startFixTimeIntervalId` value falls between `enProfile_startAfterTimeIntervalId` and `enProfile_startBeforeTimeIntervalId` values, and each `en_fix` value falls between the values of `en_low` and `en_high`.

Discussion

As seen in Figure 5.3, the multi-dimensional database representation of flex-offers and flex-offer assignments is challenging as several fact tables are involved to represent flex-offer and flex-offer assignment entities. Consider the fact table `F_flexOffer` which is the central table for representation of flex-offer and flex-offer assignment pairs. This fact table is, however, also used as a dimension table in the sense that each fact has a unique ID such that `F_enProfileInterval` and `F_aggregationMeta` can reference `F_flexOffer` and in effect store facts about facts. Unlike traditional DW schemas, we thus have non-atomic composed facts.

Example 5.7 (Flex-offers as non-atomic composed facts). Consider

`F_flexOffer` and `F_enProfileInterval` fact tables. An energy profile interval (slice) always belongs to a flex-offer and any meaningful flex-offer has an energy profile interval, as a flex-offer for zero consumption/production at an undefined point in time is hardly interesting. In this context, it can be discussed what a fact actually is. It could be argued that a single fact is represented by a single row in `F_flexOffer` and many rows in `F_enProfileInterval`.

As pointed out above, we could alternatively have modelled this by using arrays in `F_flexOffer` to hold the measures that currently are represented in `F_enProfileInterval`. This would, however, make it more cumbersome to compare different measures (e.g., `en_low` with the minimum energy requirement to `en_fix` with the assigned energy) as the interval position currently represented by `intervalNr` only would be implicitly represented by the position in the array. The denormalized variant (with a fact in `F_flexOffer` for each profile interval) would increase redundancy dramatically.

The MIRABEL DW represents facts for both non-aggregated and aggregated flex-offers in a unified way. The aggregation is unlike the traditional aggregation, since the parent flex-offer contains other flex-offers that can be shifted within the parent flex-offer. We, therefore, call the contained flex-offers *shiftable child facts*.

5.4 Modelling of Time Series

In MIRABEL DW, time series are represented by the tables shown in Figure 5.4.

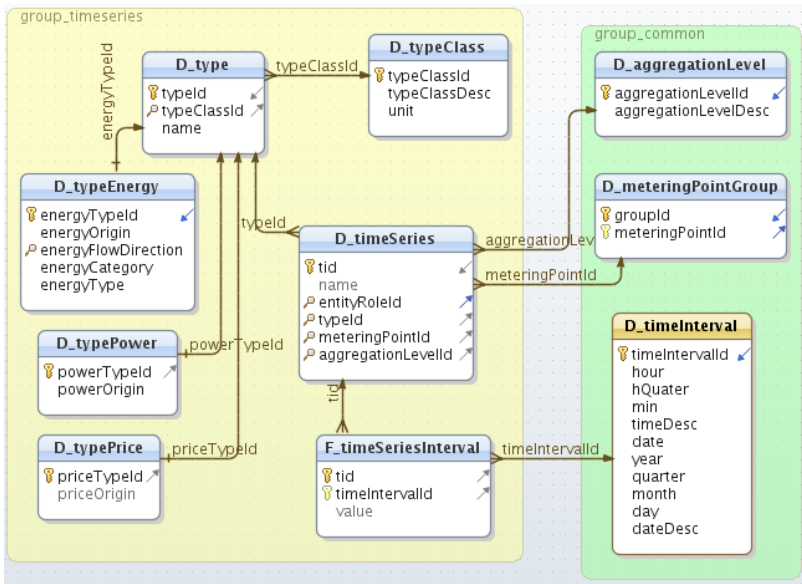


Figure 5.4: Tables for representing time series

It is necessary to be able to represent time series of energy (see Section 3.6) as well as other data types [39] such as *power* and *price*. To represent these general classes, we use the `D_typeClass` dimension table. Apart from its surrogate key, it has the attribute `typeClassDesc` which holds a textual description of the time series type (such as “Energy”), and the attribute `unit` which holds the unit of measurements (such as “kWh”). Instances of the general classes are represented in the table `D_type`. It references `D_typeClass` to represent the hierarchy between time series types and time series type classes.

Example 5.8 (RES production as a special type of energy). “*Energy-Metered-Production-RES-Wind*” is an instance of the “Energy” class. Such type is stored in `D_type` while referencing “Energy” in `D_typeClass` for a type class.

For different types of time series, it is, however, necessary to store different information. Therefore, we introduce the tables `D_typeEnergy`, `D_typePower`, and `D_typePrice` to hold the attributes that are relevant for the different types. These tables supplement, but cannot replace, `D_type`. The reason is that we need a single table to reference

from `D_timeSeries` to represent the type of the time series in question. Thus `D_type` is referenced from `D_timeSeries`, but the special attributes for an energy time series are represented in `D_typeEnergy`. The latter table has columns to describe the origin of the energy (e.g. “Metered” or “Forecasted”), the flow direction (i.e., if it is production or consumption), the category (e.g., energy from renewable energy sources), and the type of energy (e.g. “Wind”). The design is likely to evolve in the future. For example, there is a traditional hierarchy where types roll up into categories that roll up into flow directions. A more advanced hierarchy is, however, needed to represent hybrid energy types like “At least 90% energy from renewable energy sources and the rest produced from coal”.

`D_timeSeries` holds a single entry for the entire time series. For each represented time series, there is a unique ID `tid` and further a name may be given. Further, `D_timeSeries` references `D_type` (as previously described), `D_aggregationLevel` to represent the level of aggregation of the time series, and `D_meteringPointGroup` to represent which meters the time series describes. Thus, `D_timeSeries` is mainly used to relate different dimension values that describe the represented time series. The values of the time series are, however, represented in the fact table `F_timeSeriesInterval`. This table references `D_timeSeries` to identify a time series (a value belongs to) and `D_timeInterval` to identify the time instant when the value occurred. Finally, the table holds the value itself as a measure. A fact thus exists for each value in each time series. It can, however, also be argued that a fact consists of what it represented in `F_timeSeriesInterval` *and* what is represented in `D_timeSeries` which – apart from a possible name – only points out to other dimensions.

Discussion

In our modelling of time series, the schema is neither a traditional star schema nor a snowflake schema.

One reason for this is that the representation of time series, similarly to flex-offers, also leads to non-atomic composed facts where one fact can be considered to be made up of parts in different tables (`D_timeSeries` and `F_timeSeriesInterval`). Actually, an alternative design is to merge `F_timeSeriesInterval` into `D_timeSeries` such that the values instead are represented in an array, meaning that a single time interval (and all its values) only would result in one fact. Yet another alternative is to merge `D_timeSeries` and `F_timeSeriesInterval` and have a row for each value in a time series. There are thus different possible ways to represent the complex sequence-facts arising from time series. We choose the model in Figure 5.4 since it both reduces complexity (compared to the first alternative where two arrays must be processed to find the value for a given time instant) and redundancy (compared to the second alternative where there is very wide fact for each value in the time series). These three modelling alternatives are experimentally evaluated later in Section 5.6.

Another reason is the support for different types of time series for which different

5.5 Full Data Warehouse Schema

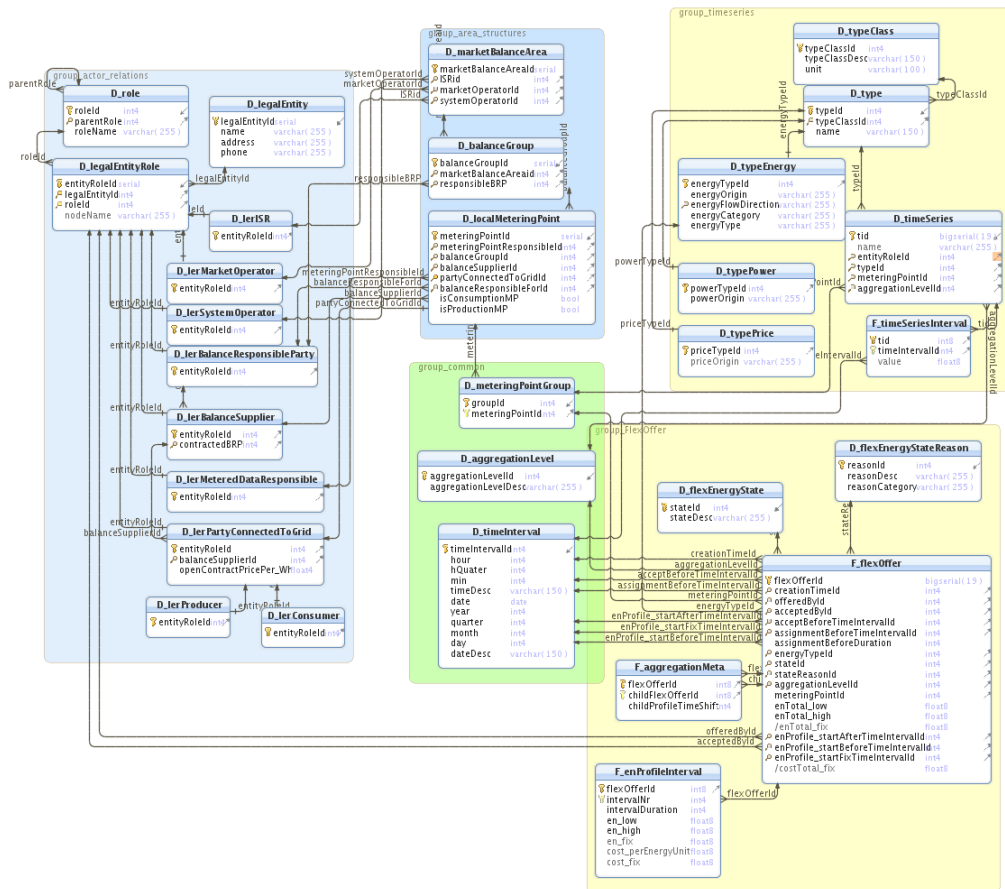


Figure 5.5: The full schema for MIRABEL DW

To summarize the previous descriptions, the full schema for MIRABEL DW is shown

in Figure 5.5. The schema is general enough to hold all the data that is defined in the MIRABEL specification [39]. Specifically, the schema captures the roles that were presented in Section 2.1 as well as additional roles from the Harmonised Model [27]. The schema also includes actors (legal entities) together with “actor configurations” specifying the set of roles an actor takes. Further, the schema captures flex-offers, flex-offer assignments, and different kinds of time series as complex multi-dimensional facts.

The schema can hold flexibility and prescription models of the types *unit*, *multi-unit*, *BG*, and *MBA*, which are composed of flex-offers, flex-offer assignments, and time series, as mentioned in Section 3.6. The following four examples point specific MIRABEL DW tables used to store each of these models.

Example 5.9 (Representation of the *unit* models). *The instances of the unit flexibility and prescription models are a flex-offer and a flex-offer assignment, respectively. A single fact in $F_flexOffer$ and multiple facts in $F_enProfileInterval$ represent these two model instances.*

Example 5.10 (Representation of the *multi-unit* models). *The instances of the multi-unit flexibility and prescription models are a set of flex-offers and a set of flex-offer assignments, respectively. Multiple facts in $F_flexOffer$ and $F_enProfileInterval$ represent both these two model instances.*

Example 5.11 (Representation of the *BG* models). *The instances of the BG flexibility and prescription models are a set of time series and flex-offers and a set of flex-offer assignments, respectively. Multiple records in $D_timeseries$, $F_timeSeriesInterval$, $F_flexOffer$, and $F_enProfileInterval$ represent these two model instances.*

Example 5.12 (Representation of the *MBA* models). *The instances of the MBA flexibility and prescription models are the pair of a time series and a set of flex-offer and a set of flex-offer assignments, respectively. A single record in $D_timeseries$ and multiple records in $F_timeSeriesInterval$, $F_flexOffer$, and $F_enProfileInterval$ represent these two model instances.*

An agent hold data (incl., flexibility and prescription models) of its own role as well as roles it interchanges data with. The following four examples elaborate the types of data managed by agents in different roles in the MIRABEL use-case.

Example 5.13 (The data of a prosumer agent). *An agent in the prosumer role manages data that includes its own non-aggregated individual flex-offers and flex-offer assignments (unit models).*

Example 5.14 (The data of an aggregator agent). *An agent in the aggregator role manages data exchanged with prosumers (unit models) and data exchanged with a BRP (multi-unit models). The data includes both non-aggregated and aggregated flex-offers and flex-offer assignments.*

Example 5.15 (The data of a BRP agent). *An agent in the BRP role manages data exchanged with aggregator agents (multi-unit models), locally produced data (BG models), and data exchanged with TSO agents (multi-unit models). The data includes a set of time series (with forecasted or metered values) and a set of aggregated flex-offers and flex-offer assignments.*

Example 5.16 (The data of a TSO agent). *An agent in the TSO role manages data exchanged with BRP agents (multi-unit models) and locally produced data (MBA models). The data includes a time series (with Δ energy values) and a sets of aggregated flex-offers and flex-offer assignments.*

In summary, the MIRABEL DW schema is generic and can be used by MIRABEL ICT system agents taking different roles. We now consider and experimentally evaluate alternative flex-offer and time series modelling strategies.

5.6 Performance Study

In this section, we consider three different MIRABEL DW schema alternatives and compare them in terms of the performance of typical queries from the MIRABEL use-case. We now overview the schema alternatives, then present the experimental setup, and finally present experiments and the results of the experiments involving queries on either flex-offers, flex-offer assignments, or time series.

5.6.1 The MIRABEL DW schema alternatives

We consider three MIRABEL DW schema alternatives: **MDW**, **Denorm**, and **Array**.

MDW is the (unchanged) schema that was described in Sections 5.2–5.5.

Denorm is the denormalized variant of MIRABEL DW. In the variant, `F_flexOffer` and `F_enProfileInterval` are joined and so are `F_timeSeriesInterval` and `D_timeSeries`. To make a typical fact table for time series, the “name varchar” attribute is replaced with the foreign key of the integer type.

Array is an array-based variant of MIRABEL DW. In this variant, `F_flexOffer` and `F_enProfileInterval` are joined and so are `F_timeSeriesInterval` and `D_timeSeries`, but now grouped on all dimension references and with measures aggregated into arrays.

5.6.2 The experimental setup

We experimentally evaluate the schema alternatives using one *stimulus query* (Q1) and four *history queries* (Q2-Q5), as defined in Section 4.3. These involve flex-offers (Q1), flex-offer assignments (Q2 and Q3), and time series (Q4 and Q5). For the experiments, we use real life data set with consumption data from 963 customers (the data originates from the MEREGIO project [41]). This gives rise to 963 (energy consumption) time series with 32.1 million time series values, and 3,1 million flex-offers. We evaluate the performance on a server with two Quad Core 1.86GHz Intel Xeon CPUs, 16 GB RAM, 4 SATA 7200RPM disks (with one dedicated to a database and a log). The DBMS is PostgreSQL 9.1 [42] and has the parameter `shared_buffers` set to 4GB, `temp_buffers` to 128MB, and `work_mem` to 96MB. All tables are “fully vacuumed” such that their disk representations only take up the needed space and do not occupy unused space. Further, the tables are “analysed” such that their statistics are up-to-date. Each query is executed once in a warm-up round and then the queries are executed in a round-robin fashion such that each query gets executed five times. We report the average execution times.

5.6.3 Experiments with flex-offers (Q1)

The first query, Q1, considers the average flexibility in flex-offers, both with respect to time and amount of energy. For the MDW variant, Q1 is defined as follows:

```
Q1:  SELECT AVG((enProfile_startBeforeTimeIntervalId -
                enProfile_startAfterTimeIntervalId) *
                (SELECT SUM((en_high - en_low) * intervalDuration)
                 FROM F_enProfileInterval i
                 WHERE i.flexOfferId = f.flexOfferId)
                )
    FROM F_flexOffer f;
```

The query considers the flexibility with respect to time, i.e., the difference between *latest start time* and *earliest start time* (see Figure 2.3a). We assume that corresponding time interval IDs (`enProfile_startBeforeTimeIntervalId` and `enProfile_startAfterTimeIntervalId`) are assigned sequentially and thus the difference between the IDs of the time intervals

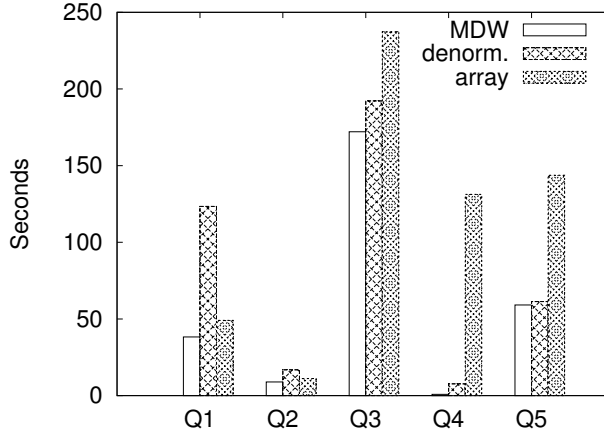


Figure 5.6: The results of the performance study

can be used to find the time flexibility. The time flexibility is multiplied with the SUM of the energy flexibility in each profile interval. The energy flexibility in a profile interval is found by multiplying the length of the profile interval (`intervalDuration`) with the difference between the maximum (`en_high`) and minimum (`en_low`) required amount of energy. This query considers the average of the combined flexibility for all flex-offers. A high number of the combined flexibility indicates much freedom in the scheduling (stimuli) while a low number shows that the considered flex-offers are not very flexible.

Results of experiments with Q1 and all three schema variants are shown in Figure 5.6(Q1). It can be seen that the MDW variant is the fastest, followed by the array variant (38.3 seconds and 49.1 seconds, respectively). These two query variants have similar query execution plans, but with arrays there are fewer rows to process. On the other hand, these rows need to have their arrays “unnested” to produce as many values as there are rows to consider in the MDW variant. When the denormalized variant is considered, there are also many rows and these rows are wide. Further, the plan is not similar to the plans for the other variants as GROUP BY is necessary with this variant. This makes the denormalized variant the slowest (123.4 seconds).

5.6.4 Experiments with flex-offer assignments (Q2 and Q3)

The second query, Q2, focuses on flex-offer assignments and, therefore, is of interest *after* the flex-offer scheduling is completed. The query gives the total amount of scheduled energy and, for the MDW variant, it is defined as follows:

```
Q2: SELECT SUM(en_fix)
      FROM F_enProfileInterval;
```

This is a simple query which, however, must read data from many rows in a realistic setting (the DBMS we use does not currently support materialized views). Results of experiments with Q2 and all three schema variants are shown in Figure 5.6 (Q2). It can be seen that the MDW variant is again the fastest (8.9 seconds) to use. Again, the array variant is the second fastest (11.1 seconds). With this variant, the arrays must again be unnested to produce the values that are available in the rows in the MDW variant. The denormalized variant uses wider rows and is the slowest (16.8 seconds).

The third query, Q3, is more complex, focuses on flex-offer assignments, and therefore is applied after scheduling has taken place. It builds a time series that, for each time interval ID, shows the amount of fixed energy. For the MDW variant, Q3 is defined as follows:

```
Q3: SELECT timeIntervalId, SUM(en_fix_part)
FROM (SELECT en_fix_part, ROW_NUMBER() OVER (PARTITION BY i.flexOfferId
      ORDER BY intervalNr) - 1 + f.enProfile_startFixTimeIntervalId
      AS timeIntervalId
FROM (SELECT flexOfferId, intervalNr, en_fix / intervalDuration
      AS en_fix_part, generate_series(1, intervalDuration)
      FROM F_enProfileInterval
      WHERE en_fix IS NOT NULL
    ) i, F_flexOffer f, D_flexEnergyState s
WHERE i.flexOfferId = f.flexOfferId AND f.stateId = s.stateId
      AND s.stateDesc = 'Assigned'
    ) AS subquery
GROUP BY timeIntervalId
ORDER BY timeIntervalId;
```

The query computes the IDs of the time intervals which are covered with flex-offer assignment's profile intervals (slices). But a profile interval has a duration (in *intervalDuration*) which defines how many time intervals the profile interval spans. Therefore, it is necessary to (evenly) distribute the profile intervals' energy amounts over one or more time intervals. To do this, one "part" row is generated for each time interval a profile interval covers by means of *generate_series*. This happens in the innermost *SELECT*. The result of this is used by the second *SELECT* which also uses the SQL window function *ROW_NUMBER* to enumerate the rows in each partition where a partition consists of the part rows for a given flex-offer assignment and is ordered by the interval numbers. Thus, the resulting row number corresponds to the number of time intervals between the assigned start time for the entire flex-offer assignment and the part represented by the row (we subtract 1 since *ROW_NUMBER* counts from 1). When we add *enProfile_startFixTimeInterval* for the flex-offer assignment, we get the ID of absolute time interval when the part executes. Finally, the outermost *SELECT* aggregates the sums of scheduled (fixed) energy amounts over all parts belonging to a given time interval.

Results of experiments with Q3 and all three schema variants are shown in Figure 5.6(Q3). It can be seen that the MDW variant remains the fastest (172.1 seconds)

while the array variant now is the slowest (237.2 seconds) even though it avoids a join. On the other hand, the array variant requires a *SELECT* clause to unnest the array and an extra use of *ROW_NUMBER* to recreate the values from *intervalNr* which only are implicitly available from the array positions. The denormalized variant (192.2 seconds) is bit slower than the MDW variant even though it avoids a join.

5.6.5 Experiments with time series (Q4 and Q5)

The fourth query, Q4, finds the balance, i.e., the difference between produced and consumed energy, for a 24 hours period. For the MDW variant, Q4 is defined as follows:

```
Q4: SELECT date, timeDesc,
        SUM(CASE energyFlowDirection WHEN 'Production' THEN value
        ELSE 0 END) AS production,
        SUM(CASE energyFlowDirection WHEN 'Consumption' THEN value
        ELSE 0 END) AS consumption
        SUM(CASE energyFlowDirection WHEN 'Production' THEN value
        WHEN 'Consumption' THEN -1 * value
        ELSE 0 END) AS balance
FROM F_timeSeriesInterval f, D_timeSeries ts, D_type ty,
     D_typeEnergy te, D_timeInterval ti
WHERE f.tid = ts.tid AND ts.typeId = ty.typeId AND te.energyTypeId =
      ty.typeId AND ti.timeIntervalId = f.timeIntervalId AND
      te.energyOrigin = 'Metered' AND ti.date = '2011-06-01'
GROUP BY ti.timeIntervalId
ORDER BY ti.timeIntervalId;
```

The query Q4 is an example where we use the special attributes that only apply to some time series. In this example, we consider consumed and produced energy, and we thus use *energyFlowDirection* and *energyOrigin* which only exist for energy time series. The query sums the production values, consumption values, and the difference between them for each time interval that belongs to a given date (2011-06-01 in this case).

Results of experiments with Q4 and all three schema variants are shown in Figure 5.6(Q4). It can be seen that the MDW variant is significantly faster (0.8 seconds) than the others. The denormalized variant which avoids a join uses an order of magnitude more time (7.7 seconds). The array variant is by far the slowest (131.9 seconds) as there is no index on *timeIntervalId* which is an array. Thus all rows must be processed and have their rows unnested to perform a join with *D_timeInterval*.

Our last query, Q5, finds those time series where the average energy usage grouped on hours exceeds the average energy usage for the hour with more than 25% at least 10 times. For the MDW variant, Q5 is defined as follows:

```
Q5: WITH indavguse AS (
      SELECT tid, hour, COUNT(value) AS indcnt, AVG(value) AS indavg
      FROM F_timeSeriesInterval NATURAL JOIN D_timeInterval
      GROUP BY tid, hour
    ),
```

```

totavguse AS (
    SELECT hour, SUM(indcnt * indavg) / SUM(indcnt) AS totavg
    FROM indavguse
    GROUP BY hour
),
overuse AS (
    SELECT tid, t.hour, indavg, totavg,
           COUNT(*) OVER (PARTITION BY tid) AS cnt
    FROM totavguse t, indavguse i
    WHERE t.hour = i.hour AND indavg >= 1.25 * totavg
)
SELECT tid, cnt, hour, indavg, totavg
FROM overuse
WHERE cnt > 10
ORDER BY tid, hour;

```

The query has Common Table Expressions (CTEs) in the WITH part. In the first CTE, *indavguse*, we compute a (temporary) table with an average hourly energy usage for each time series. The result is used again to compute the second CTE, *totavguse*, where we find the average energy use per hour among all time series (we could join *F_timeSeriesInterval* and *D_timeInterval* again, but it is faster to reuse the result of the previously computed CTE). In the third CTE, *overuse*, we join the results of the two previous CTEs to find the IDs of time series and the hours from *indavguse* where the consumption is 25% higher than the general hourly average consumption found in *totavguse*. Further, we use *COUNT* as a window function to count how many such hours we find for a given time series. Finally, we select the ID of the time series, the count of hours with an average energy usage at least 25% higher than the average, and the consumption in the last *SELECT* clause.

Results of experiments with Q5 and all three schema variants are shown in Figure 5.6(Q5). It can be seen that the MDW and denormalized variants perform similarly (59.1 and 61.3 seconds, respectively). The queries involve the same number of rows and are identical apart from that the denormalized variant uses a wider table. For the array variant, the first CTE has to unnest two arrays and the query takes longer time (143.8 seconds).

5.6.6 Summary

To summarize, the MDW variant performs the best for all queries compared to the denormalized and array-based variants. Another interesting thing to consider is the disk space usage. The tables *F_flexOffer*, *F_enProfileInterval*, *F_timeSeriesInterval*, and *D_timeSeries* take up 4.1 GB in the MDW variant (not counting indexes). Their alternative representations take up 7.0 GB in the denormalized variant and 1.9 GB in the array variant, respectively. It is notable how little space the array variant uses compared to the other variants due to its fewer number of rows (and thus fewer space-consuming row headers). Overall, the MDW variant is a good choice for the MIRABEL DW considering both its

performance and space requirements.

5.7 Related Work

In this section, we review related work on the modelling of energy-related entities, including actors and time series.

In the energy sector, there is number of standardized data models used to represent the major objects in an electric utility enterprise [43] as well as to define administrative data internally interchanged between European electricity markets [27, 44]. These models focus on various aspects of energy trading and physical electricity delivery, and specify (1) components of the power system at the electrical level, (2) actors and roles involved in the energy trading, (3) relationships and data exchange between those entities. These models are used as a basis for the MIRABEL data model [39], which further enriches them with the concept of flexible consumption and production (flex-offers). All these models, however, focus on semantic rather than the storage or the management of energy-related entities. By focusing on the two most important entities in MIRABEL-time series and flex-offers, this chapter, on the other hand, presents data representation models (MDW, Denorm, Array) that offer a convenient storage and a good performance of analytical queries for the instances of flexibility and prescription models defined as time series and flex-offer entities.

Our proposed MIRABEL DW schema is the first to address with the storage of flex-offers, but there are previous works which focus the time series warehousing, e.g. UML-based modelling of time-series in DWs [45], and temporal aggregation of multi-dimensional data [46], and temporal DWs exploiting research results from the field of temporal databases [47]. Our modelling of different time-series types has similarities with Bauer et al's work [48]. They discuss "locally valid dimensional attributes" whose existence depends on values of dimensional elements. This is the case, e.g., for our attribute `energyType` which only exists if the `D_type` value represents an energy time-series. The problem of representing all these attributes in a single dimension table (as in a typical star schema) is that there will be many NULLs in the held data. Bauer et al. propose to have separate tables with the specific attributes and then create views "on top" of these with common attributes as well as textual values showing the name of the relation the data comes from which can be used for hierarchical classification. In contrast, we use tables (and not views) for the common attributes of a dimension and then represent special attributes that only exist for some dimensional values in other tables that reference the table with the common attributes. This makes it possible to declare foreign keys to the dimension table with the common attributes and also declare indexes and constraints on these tables. Bauer et al. also propose to use table inheritance to represent such cases. This would also be possible in our DBMS [42], but constraints cannot be enforced on child tables then.

For the MIRABEL DW schema, we considered different representations of profile

intervals and time series intervals. These can be considered as facts with multi-valued measures. The latter case also has a many-to-many relationship between the time series facts and the time interval dimension. Previous work [49] has considered many-to-many relationships between fact tables and dimension tables. Our denormalized representation is similar to one of the methods of [49] whereas our other approaches with fact tables referencing other fact tables and measure values in arrays, respectively, are different.

5.8 Summary and Discussion

In this chapter, we have presented a DW schema for managing the complex MIRABEL-specific entities, which include actors playing roles, flex-offers, flex-offer assignments, and different types of time series. The schema has a number of interesting complexities such as facts about facts and composed non-atomic facts. We have considered different alternatives for the schema modelling using denormalization and arrays, respectively, but based on the performance and space usage, the chosen design is favourable.

Flex-offers, flex-offer assignments, and time series are the core building blocks of the *unit*, *multi-unit*, *BG*, and *MBA* flexibility and prescription models (see Section 3.6). Therefore, the schema is generic and can be used for storing instances of these flexibility and prescription models. Consequently, MIRABEL ICT system agents (actors) in the *prosumer*, *aggregator*, *BRP*, and *TSO* roles can use the schema to generate their own local databases (data warehouses) for the storage of agent-specific flexibility and prescription models and related data. As seen in the case of MIRABEL DW, flexibility and prescription models are, in fact, complex multi-dimensional structures spanning across multiple tables in a database (warehouse).

Other similar instances of PRESCRIPTIVECPS may reuse the modelling strategies and/or schema elements applied for the MIRABEL DW. The actor-role modelling is of a particular interest, as agents of other instances of PRESCRIPTIVECPS are also expected to deal with multiple actors, potentially, in several roles. Furthermore, a time series is a common mean for representing various measurements from real-world. Therefore, the use of time series is also expected in other similar instances of PRESCRIPTIVECPS. Finally, the modelling of flex-offers and flex-offer assignments as multi-dimensional objects where flex-offer and flex-offer assignment pairs are stored as joint non-atomic composed facts can be reused while designing databases for other instances of PRESCRIPTIVECPS, also dealing with complex multi-dimensional flexibility and prescription models.

Chapter 6

Flexibility Model Aggregation

In this chapter, we consider PRESCRIPTIVECPS agents in the *aggregator* role and show how flexibility models can be aggregated using a lossy *reduce* operation (see Section 3.3). Note, lossless *reduce* variants are straightforward and less interesting. For specificity, we consider the lossy *reduce*^A operation (used by an aggregator agent, see Section 3.6), which aggregates a set of *unit* flexibility models to a single *multi-unit* flexibility model. As *unit* and *multi-unit* flexibility models are represented solely by flex-offers, the *reduce*^A operation essentially maps N flex-offers into M flex-offers such that $M \leq N$. In this chapter, we present a three-phase flex-offer aggregation technique, which allows controlling a trade-off between the loss of stimulus (Requirement 5) and the preservation of response (Requirement 6). Additionally, we discuss an incremental aggregation technique, allowing to efficiently accommodate input flex-offer changes without the need to recompute (re-aggregate) output flex-offers from scratch. Extensive experiments show that the proposed techniques (1) provide good performance (Requirement 1) while still satisfying the strict correctness requirement (Requirement 2), and (2) allow improving flex-offer scheduling (*reduce*^B) result by up to 20 times, compared to the case when no aggregation is performed. We also discuss the corresponding *multi-unit* prescription model disaggregation operation *map*^A (used by an aggregator agent, see Section 3.6). We conclude the chapter by reviewing the related work and discussing flexibility model aggregation in other similar instances of PrescriptiveCPS. The content of this chapter is based on Publications [3, 4].

6.1 Flexibility Model Aggregation in MIRABEL

As explained in Section 4.1, the planning component of a PRESCRIPTIVECPS agent software system includes one or more realizations of the *reduce* operation (see Section 3.3), which aggregates a set of (smaller) flexibility models into a single (larger) flexibility

model. In the general case, the realization of *reduce* operation is specific to flexibility models used as input and output, and the required level of stimulus loss (Requirement 5) and response preservation (Requirement 6).

In the MIRABEL use-case, these are three flexibility model aggregation functions (realizations), namely $reduce^A$, $reduce^B$, and $reduce^T$ used by aggregator, BRP, and TSO agents, respectively. Here, $reduce^B$ and $reduce^T$ are lossless aggregation functions as they simply “fuse” input flexibility models into *BG* and *MBA* flexibility models, respectively (less interesting). On the other hand, $reduce^A$ is a lossy function, aggregating the set of *unit* flexibility models into a single *multi-unit* flexibility model. As *unit* and *multi-unit* flexibility models are represented solely by flex-offers, the $reduce^A$ aggregates N flex-offers into M flex-offers such that $M \leq N$. In the MIRABEL use-case, $reduce^A$ plays a crucial role as it allows substantially reducing the number of flex-offers that need to be considered by a BRP agent during the (computationally expensive) scheduling of flex-offers. In this chapter, consequently, we focus on and propose approaches for the $reduce^A$ operation. We also show that prescription models built according to $reduce^A$ outputs (flexibility models) can be correctly disaggregated with map^A (Requirement 2).

As will be shown later, the aggregation of flex-offers is a non-trivial problem as there are many possible ways to combine just two flex-offers. To address the problem, we present two techniques for $reduce^A$ – an *instant* and an *incremental*. These two realize the N-to-M lossy aggregation of flex-offers and, based on the provided aggregation parameters, allows controlling the trade-off between the loss of stimulus (Requirement 5) and the preservation of response (Requirement 6). The techniques partition the set of flex-offers into disjoint groups of similar flex-offers. This partitioning is performed in two steps - grid-based *grouping* and *bin-packing*. The grouping of flex-offers ensures that flex-offers in a group are sufficiently similar (in terms of chosen flexibility attributes). The bin-packing ensures that the groups themselves conform to the given (aggregate) criteria. After bin-packing, an (N-to-1) aggregation operator is applied to merge similar flex-offers into aggregated flex-offers.

We now define the flex-offer aggregation problem and present the *instant* and *incremental* aggregation techniques.

6.2 Problem of Flex-offer Aggregation

We now formalize the problem of flex-offer aggregation (and disaggregation). Our formalization includes (1) definitions of a flex-offer and a flex-offer assignment, (2) two measures to quantify the amount of stimuli and responses, and (3) a correctness requirement for the $reduce^A$ (aggregation) and map^A (disaggregation) functions.

A flex-offer from MIRABEL can be seen as a multidimensional object capturing two essential components: (1) a *start-time interval* and (2) a *data profile* defined as a sequence of consecutive *slices* each defined by (a) its duration (extent in time) and

(b) the minimum and maximum energy amount bounds. We can formally define a flex-offer as follows:

Definition 6.1. A flex-offer f is a tuple $f = (T(f), P(f))$, where $T(f)$ is the start-time interval and $P(f)$ is the data profile. Here, $T(f) = [t_{es}, t_{ls}]$, where t_{es} and t_{ls} are the earliest start time and latest start time, respectively. The $P(f) = s^{(1)}, \dots, s^{(m)}$, where a slice $s^{(i)}$ is a tuple $(d, [a_{min}, a_{max}])$, $d \geq 0$ is a slice-duration, and $[a_{min}, a_{max}]$ is a continuous range of energy amount. We use the term profile duration to denote $p_{dur}(f) = \sum_{s \in P(f)} s.d$ and the terms earliest end time and latest end time to denote $t_{ee}(f) = T(f).t_{es} + p_{dur}(f)$ and $t_{le} = T(f).t_{ls} + p_{dur}(f)$, respectively.

The corresponding flex-offer assignment (prescription model) for a given flex-offer (flexibility model) can be defined as follows:

Definition 6.2. A flex-offer assignment f_x of a given flex-offer $f = ([t_{es}, t_{ls}], s^{(1)}, \dots, s^{(m)})$ is another flex-offer $f_x = ([t_s, t_s], s_x^{(1)}, \dots, s_x^{(m)})$ such that $t_{es} \leq t_s \leq t_{ls}$ and $\forall i = 1..m : s^{(i)}.d = s_x^{(i)}.d \wedge s^{(i)}.a_{min} \leq s_x^{(i)}.a_{min} = s_x^{(i)}.a_{max} \leq s^{(i)}.a_{max}$. We refer to t_s as the start-time of the flex-offer assignment f_x , and use $s_e(s_x^{(i)})$ to denote an (absolute) end time of a slice $s_x^{(i)}$, where $s_e(s_x^{(i)}) = f_x.t_s + \sum_{j=1..i} f_x.s_x^{(j)}.d$.

Example 6.1 (Flex-offer and flex-offer assignment example). Figure 6.1 depicts the example of a flex-offer having a profile with four slices: $s^{(1)}$, $s^{(2)}$, $s^{(3)}$, and $s^{(4)}$. Every slice is represented by a bar in the figure. The area of the light-shaded bar represents the minimum amount value (a_{min}) and the combined area of the light- and dark-shaded bars represents the maximum amount value (a_{max}). There is an infinite number of possible flex-offer assignments (instances of a flex-offer). One possible flex-offer assignment is shown as the dotted line in Figure 6.1.

To be able to quantify stimulus and response losses, we need to measure amounts of *stimulus* and *response* before and after aggregation with *reduce*^A. The inputs and outputs of *reduce*^A are sets of flex-offers (multiple *unit* models as input and a single *multi-unit* model as output), therefore, the measures of *stimulus* and *response* for sets

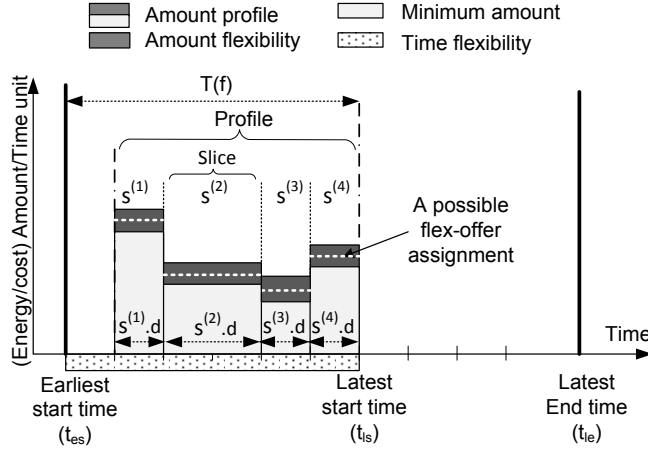


Figure 6.1: An example of a flex-offer

of flex-offers are needed. However, due to continuous ranges used in a flex-offer, it is impossible to count the (infinite) number of stimuli and responses represented by even a single flex-offer. Instead, we can define “indirect” simplified measures allowing to quantify the total amount of *stimulus* and *response*, represented either by a single flex-offer or a set of flex-offers. To define such measures, we make a further simplification by assuming that each flex-offer stimulus maps into a distinct response (i.e., the response function R is injective) and then use a common measure to quantify both *stimulus* and *response*. We use the *flex-offer count*, $\text{count}(F) = |F|$, as the simplest measure of *stimulus* and/or *response*, as exemplified below.

Example 6.2 (Flex-offer count as a measure of stimulus and response).

Given a flex-offer set $F = \{f_1, f_2, f_3\}$ (a multi-unit flexibility model), the amount of stimuli and/or responses represented by F is 3 (i.e., $\text{count}(F) = 3$), each flex-offer representing 1 unit of stimulus and/or response.

As more advanced measure of *stimulus* and/or *response* (among many others), we define a measure of *flexibility*, which we first define for a (single) flex-offer and then generalize it for the set of flex-offers.

Definition 6.3. The flexibility of a (single) flex-offer f is the product of start-time flexibility and a profile flexibility, i.e., $\text{flex}(f) = \text{flex}_T(f) \cdot \text{flex}_P(f)$. Here, start-time flexibility, $\text{flex}_T(f)$, is the difference between the latest and earliest start time,

i.e., $\text{flex}_T(f) = f.t_{ls} - f.t_{es}$. Similarly, the profile flexibility, $\text{flex}_P(f)$, is the sum of the products of durations and the amount differences of all slices in the profile, i.e., $\text{flex}_P(f) = \sum_{s \in P(f)} (s.d) \cdot (s.a_{max} - s.a_{min})$.

Definition 6.4. The flexibility of a flex-offer set F is the sum of flexibilities of every individual flex-offer, i.e., $\text{flex}(F) = \sum_{f \in F} \text{flex}(f)$.

Example 6.3 (Flexibility as a measure of stimulus and response).

Consider a flex-offer $f = ([2, 7], s^{(1)}, s^{(2)})$ where $s^{(1)} = (1, [10, 20])$ and $s^{(2)} = (3, [6, 10])$. The start-time flexibility, $\text{flex}_T(f)$, is equal to $7 - 2 = 5$. The profile flexibility, $\text{flex}_P(f)$, is equal to $1 \cdot (20 - 10) + 3 \cdot (10 - 6) = 22$. Hence, the flexibility of f is 110. Given two identical flex-offers $f_1 = f_2 = f$, the flexibility of $\{f_1, f_2\}$ is 220. Note, the flexibility of a corresponding flex-offer assignment and flex-offer assignment set is always equal to zero, i.e., they are always “inflexible”.

Using the notations above, the reduce^A and map^A – the flex-offer aggregation and flex-offer assignment disaggregation functions from Section 3.6 – are defined as follows:

Definition 6.5. Let reduce^A be a so-called lossy flex-offer aggregation function which takes a set of flex-offers F (many unit flexibility models) and produces a set of flex-offers A (a single multi-unit flexibility model) such that $\text{flex}(A) \leq \text{flex}(F)$ (and $\text{count}(A) \leq \text{count}(F)$). Every $f_a \in A$ is called an aggregated flex-offer.

Definition 6.6. Let map^A be a so-called flex-offer disaggregation function which takes a set of flex-offer assignments A_x (a single multi-unit prescription model) and produces a set of flex-offer assignments F_x (many unit prescription models).

We term the evaluation of the functions $reduce^A$ and map^A as flex-offer *aggregation* and *disaggregation*, respectively. There exist many $reduce^A$ and map^A function pairs, but we are primarily interested in such function pair that allows for the consistent views of a physical system and prescriptions (Requirement 2). This correctness requirement for the pair of $reduce^A$ and map^A can be formalized as follows:

Definition 6.7. Given $A = reduce^A(F)$ and $F_x = map(A_x)$ where $\forall f \in F \Leftrightarrow \exists f^x \in F_x$ and $\forall f_a \in A \Leftrightarrow \exists f_a^x \in A_x$, the energy balance at aggregated and non-aggregated levels must be ensured, i.e., for all time units $T = 0, 1, 2, \dots$ the following equality must hold:

$$\sum_{t=0}^T [s.a_{min} | \forall s \in P(f_a^x), \forall f_a^x \in A_x, s_e(s) \leq t] = \sum_{t=0}^T [s.a_{min} | \forall s \in P(f_x), \forall f_x \in F_x, s_e(s) \leq t].$$

Due to this correctness requirement, disaggregation is, however, not always possible for any arbitrary $reduce^A$ function. Depending on whether a disaggregation is possible or not without violating this constraint, we distinguish two types of flex-offer aggregation: *conservative* and *greedy*, respectively.

The *greedy aggregation* produces aggregated flex-offers which might define more (time and profile) flexibility compared to the original flex-offers ($flex(A) > flex(F)$). Obviously, the corresponding flex-offer assignments (flex-offer instances) might not be disaggregated using map^A without violating the correctness requirement. The correctness requirement might also be violated due to inconsistencies of flex-offers and flex-offer assignments occurring due, e.g., communication delays in PRESCRIPTIVECPS. To disaggregate flex-offers in such cases or the case of the greedy $reduce^T$, the $solve_M$ operation is used (see Section 3.3). It performs disaggregation by solving an optimization problem to resolve inconsistencies between adjacent aggregation levels (to an extent possible). Greedy $reduce^T$ and $solve_M$ are out of the scope of this thesis (future work).

The *conservative aggregation*, on the other hand, produces aggregated flex-offers which always define less (or equal) flexibility compared to the original flex-offers ($flex(A) \leq flex(F)$). It is, therefore, possible to disaggregate flex-offer assignments using map^A without violating the correctness constraint. We now focus exclusively on a conservative aggregation and present several flex-offer aggregation techniques allowing for a deterministic disaggregation (requiring no optimization problem solving), while satisfying the correctness requirement.

6.3 N-to-1 Aggregation Technique

In this section, we first propose a basic N-to-1 flex-offer aggregation algorithm and explain how to generalize for it a large set of flex-offers. Additionally, we explain how the disaggregation can be performed while satisfying the correctness requirement.

The aggregation of flex-offers is somewhat similar to an aggregation (addition) of (overlapping) time series, but it is (substantially) more complicated as there are many ways to align (shift) flex-offer profiles within their allowed start-time flexibility intervals before adding values of corresponding flex-offer slices. For three flex-offers, this issue is demonstrated in the following example:

Example 6.4 (Aggregation of three flex-offers). *Consider aggregating three flex-offers f_1 , f_2 , and f_3 with start-time flexibility values (t_{ls} - t_{es}) equal to five, three, and four respectively. Thus, we have 60 ($5 \cdot 3 \cdot 4$) different profile start-time combinations, each of them realizing a different aggregated flex-offer. Three possible profile start-time parameter combinations and the resulting aggregated flex-offers are shown in Figure 6.2(a-c).*

To overcome this issue, we can align (“fix in time”) flex-offer profiles before adding their slices, leading to the following three step procedure of aggregating a set of flex-offers F into a single aggregated flex-offer f_a :

1. For each $f \in F$, we choose a so-called *profile shift* s_f such that $f.t_{es} \leq s_f \leq f.t_{ls}$. We refer to the choice of profile shift as *profile alignment*.
2. For f_a , set the start-time flexibility interval such that $f_a.t_{es} = \min_{f \in F}(s_f)$ and $f_a.t_{ls} = f_a.t_{es} + \min_{f \in F}(f.t_{ls} - s_f)$.
3. For f_a , build a profile $P(f_a)$ by summing the corresponding (a_{min} and a_{max}) amounts for each slice across all profiles, which are specially pre-partitioned (segmented) to make slices adjacent and with equal durations across all profiles.

There are many ways to align profiles (by choosing the constants s_{f_1} , s_{f_2} , ..., $s_{f_{|F|}}$). Each of these alignments determines where amounts from individual flex-offers are concentrated within the profile of f_a . We focus only on the three most important alignment options: *start-alignment*, *soft left-alignment*, and *soft right-alignment*. Start-alignment spreads out amounts throughout the time extent of all individual flex-offers, making larger amounts available as early as possible. On the contrary, soft left-/right-alignment builds shorter profiles with amounts concentrated early (left) or late (right) in the profile. The impact of these alignments is demonstrated below.

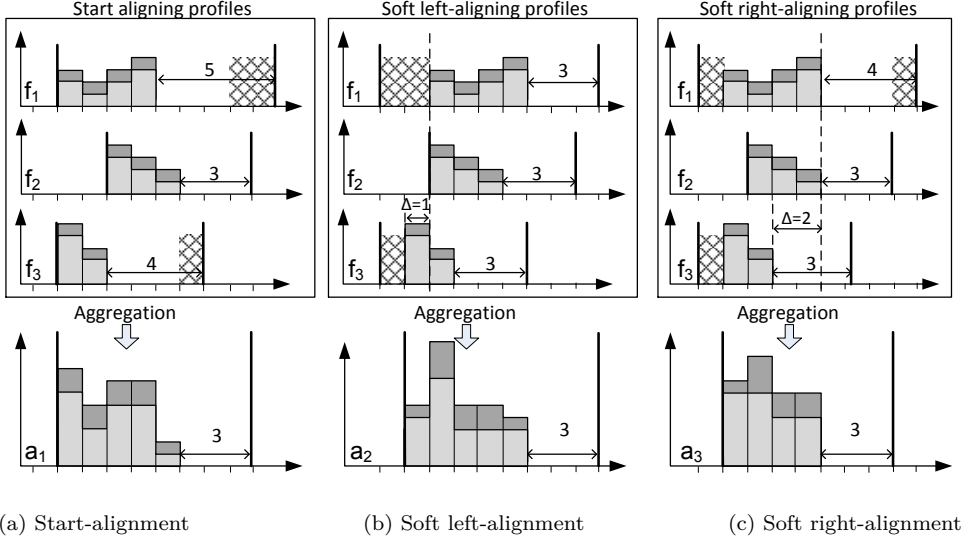


Figure 6.2: The effects of the three profile alignment options

Example 6.5 (Start-, soft left-, and soft right-alignments in MIRABEL).

In MIRABEL, start-alignment is suitable for the near real-time balancing of electricity, where energy has to be available as early as possible. Soft left/right alignment allows the consumption of anticipated wind production peaks with steep rises (left-alignment) or falls (right-alignment).

The three alignment options are illustrated in Figure 6.2. Here, the crossed area in the figure represents the amount of time flexibility that is lost due to aggregation with different profile alignments. The alignment options are elaborated below:

Start-alignment. We set $s_{f_1}, s_{f_2}, \dots, s_{f_{|F|}}$ so that $\forall f \in F : s_f = f.t_{es}$. This ensures that profiles are aligned at their respective *earliest start time* values (see f_1 and f_2 in Figure 6.2(a)).

Soft left-alignment. We set $s_{f_1}, s_{f_2}, \dots, s_{f_{|F|}}$ so that $\forall f \in F : s_f = \min(f.t_{ls} - \min_{g \in F} (g.t_{ls} - g.t_{es}), \max_{g \in F} (g.t_{es}))$. Figure 6.2(b) illustrates the effect of soft left-alignment. Here, f_1 and f_2 are left-aligned, meaning that their profile start times are equal. However, the profile of f_3 cannot be (strictly) left-aligned with respect to the profiles of f_1 and f_2 as that would shorten the remaining time flexibility range of the aggregate. f_3 lacks one time unit ($\Delta = 1$) for its profile to (strictly) left-align.

Soft right-alignment. We set $s_{f_1}, s_{f_2}, \dots, s_{f_{|F|}}$ so that $\forall f \in F : s_f = \min(f.t_{ls} - \min_{g \in F} (g.t_{ls} - g.t_{es}), \max_{g \in F} (g.t_{es} + p_{dur}(g)) - p_{dur}(f))$. Figure 6.2(c) illustrates the

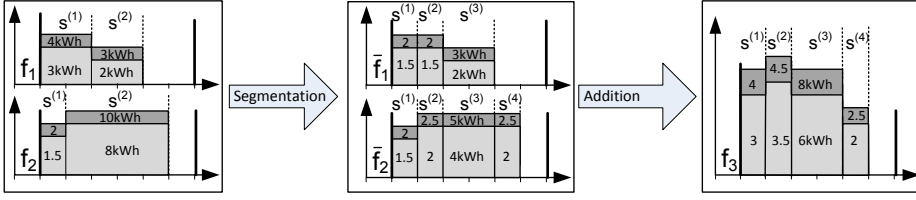


Figure 6.3: The effects of the profile segmentation and slice addition

effect of soft right-alignment. Here, f_1 and f_2 are right-aligned, meaning that their profiles align at the right hand side (i.e., have equal $t_{es} + p_{dur}$ values). However, the profile of f_3 cannot be (strictly) right-aligned with respect to the profiles of f_1 and f_2 as this would shorten the remaining time flexibility range of the aggregate.

After the alignment (step 1), the time flexibility interval is computed for the aggregated flex-offer (step 2). As illustrated in Figure 6.2(a-c), for all three alignment options, the time flexibility of f_a is equal to that of the flex-offer with the smallest time flexibility in the set F , i.e., $f_a.t_{ls} - f_a.t_{es} = \min_{f \in F} (f.t_{ls} - f.t_{es})$. However, other types of alignment, e.g., *strict left* or *strict right* where all profiles are forced to align at the left or right hand side, might reduce the time flexibility of the aggregated flex-offer.

Finally, the minimum and maximum energy amounts of adjacent slices in the aligned profiles are summed to construct the profile of the aggregated flex-offer (step 3). If adjacent slices at any time unit have different durations, those slices are partitioned to unify their durations. During the partitioning, minimum and maximum energy amounts are distributed proportionally to the duration of each divided slice. This step is called *segmentation*. The segmentation step reduces the profile flexibility, $flex_P(f)$, as it imposes more restrictions on the energy amount for each divided segment.

Example 6.6 (Flex-offer profile segmentation). Consider the slice $s^{(1)}$ of f_1 in Figure 6.3, which illustrates the segmentation for two flex-offers. Originally, the minimum energy amount is 3kWh and the maximum energy amount is 4kWh over two time units. Thus, we can supply one amount unit in the first time unit, and three units in the second time unit. However, this supply is not acceptable after dividing the slice into two equal-sized slices $s^{(1)}$ and $s^{(2)}$ with minimum and maximum amount of 1.5kWh and 2kWh, respectively.

After the segmentation, the addition of profiles is performed similarly to an addition of time series. During the addition, a_{min} and a_{max} energy amounts are added for every corresponded profile slice, as shown in Figure 6.3.

It is always possible to disaggregate flex-offer assignments of a flex-offer produced by this aggregation approach without violating the correctness requirement (see Section 6.2). Consider the following disaggregation procedure. For a given flex-offer assignment f_a^x of an aggregated flex-offer f_a , we produce the set of flex-offer assignments $\{f_1^x, f_2^x, \dots, f_{|F|}^x\}$ such that $\forall i = 1..|F| : f_i^x.t_s = s_{f_i} + (f_a^x.t_s - f_a.t_{es})$. It is always possible to fix the *start time* of every $f_i^x, i = 1..|F|$ because the time flexibility range of the aggregate f_a is computed conservatively, and the aligned profiles of $f \in F$ can always be shifted within this range (see Figure 6.2). Also, the energy amount values from every slice $s_a^x \in P(f_a^x)$ are distributed proportionally to the respective slices of $f_i^x, i = 1..|F|$ so that minimum and maximum amount constraints are respected for every $f \in F$. This can always be achieved, and consequently, for any flex-offer assignment f_a , it is always possible to build flex-offer assignments for all flex-offers in F . The correctness requirement is satisfied, as the newly built flex-offer assignments will collectively define total amounts which, at every time interval, are equal to the corresponding amounts of the initial flex-offer assignment f_a^a .

To summarize, the N-to-1 aggregation approach can be used to aggregate flex-offers in F . However, the time (and total) flexibility loss depends on a flex-offer with the smallest time flexibility in the set F . Due to this issue, much of the flexibility will be lost when aggregating flex-offers with distinct time flexibilities. To address this, we will now propose an N-to-M aggregation approach.

6.4 N-to-M Aggregation Technique

As discussed in Section 6.3, aggregating “non-similar” flex-offers results in unnecessary loss of time flexibility. This loss can be avoided, and the profile alignments can be better enforced, by carefully grouping flex-offers and thus ensuring that their time flexibility intervals overlap substantially. We now describe a so-called *N-to-M* approach to aggregate a set of flex-offers, F , to a set of aggregated flex-offers, A , while satisfying the strict correctness requirement (see Section 6.2) and allowing to control the loss of stimulus and response (Requirements 5–6). The algorithm consists of three phases: *grouping*, *bin-packing*, and *N-to-1 aggregation*:

Grouping phase. We partition the input set F into disjoint groups of similar flex-offers. Two flex-offers are grouped together if the values of user-specified grouping attributes differ by no more than user-specified tolerances. The tolerances and the associated grouping attributes are called *grouping parameters*.

Example 6.7 (Grouping parameters). Suppose that a user chooses the earliest start-time (t_{es}) as a grouping attribute and specifies an earliest start-time tolerance (EST) equals to 2 time units. Then, during the grouping phase the flex-offers f_1, f_2, \dots , and f_5 , shown in Figure 6.4, are assembled in two groups g_1 and g_2 . Within g_1 and g_2 flex-offers have earliest start-time values differing by no more than $EST = 2$.

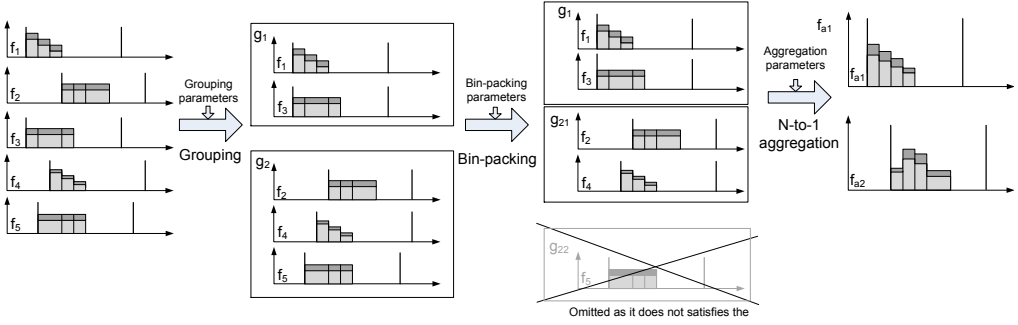


Figure 6.4: The phases of flex-offer aggregation using the N-to-M aggregation approach

Note, different grouping attributes can be chosen, e.g., earliest start-time (t_{es}), latest start-time (t_{ls}), time flexibility ($flex_T(f)$), and/or profile flexibility ($flex_P(f)$).

As shown later, the choice of grouping parameters yields a particular flex-offer count reduction (compression) and stimulus/response loss.

Bin-packing phase. This phase enforces a so-called *aggregate constraint*, which is satisfied only if the value of a certain user-specified flex-offer attribute, e.g., *total maximum amount*, is within given bounds. Each group g produced in the *grouping* phase is either passed to the next phase (if g satisfies the constraint already) or further partitioned into the minimum number of bins (groups) such that the constraint $w_{min} \leq w(b) \leq w_{max}$ is satisfied by each bin b . Here, $w(b)$ is a weight function, e.g., $w(b) = |b|$, and w_{max} and w_{min} are the upper and lower bounds. We refer to w_{min} , w_{max} , and w as *bin-packing parameters*. Such bin-packing is effective when aggregating many similar (or equal) flex-offers, as demonstrated in the example below:

Example 6.8 (Aggregate constraint over a set of similar flex-offers).

Consider many similar (or equal) flex-offers. These cannot be grouped into disjoint groups in the grouping phase and might result into aggregated flex-offers that are “too-large” in terms of energy amount and thus violate the rules of the power grid, BRP, and the energy market, or, simply, lose too much of stimulus and response. By enforcing the aggregate constraint in the bin-packing phase (using bin-packing parameters), “proper” groups are built, resulting into “proper” aggregated flex-offers, e.g., flex-offers with bounded total energy amounts or built from a bounded number of non-aggregated flex-offers.

Note that it may be impossible to satisfy a user-specified aggregate constraint for certain groups.

Example 6.9 (Aggregate constraint being too tight). Consider a group with a single flex-offer, while we impose a lower bound of two flex-offers in all groups. Such group is discarded from the output (see g_{22} in Figure 6.4), and, depending on the use-case, respective flex-offers are either: (1) excluded from the N-to-M aggregation output, or (2) aggregated with another instance of the N-to-M aggregation, but with less constraining grouping or bin-packing parameters.

N-to-1 Aggregation phase. We assemble the output set A by applying N-to-1 aggregation (see Section 6.3) for each resulting group g . The alignment option is specified as an *aggregation parameter*. Every aggregated flex-offer satisfies the aggregate constraint enforced in the bin-packing phase.

The complete N-to-M aggregation process is visualized in Figure 6.4. Here, given the initial flex-offer set $\{f_1, f_2, \dots, f_5\}$ and grouping, bin-packing, and aggregation parameters, two aggregated flex-offers, f_{a1} and f_{a2} , are produced. The grouping parameters are exemplified in Example 6.7. The bin-packing parameters require that the number of flex-offers in resulting groups is 2, i.e., $w_{min} = w_{max} = 2$, $w(g) = |g|$. In the aggregation phase, the *start-aligned* option is used. In the MIRABEL use-case, the aggregator agent will use meaningful and pre-defined *reduce^A* parameter settings, e.g., *short/long profiles* or *amount as early as possible*.

6.5 Incremental Aggregation Technique

In the MIRABEL use-case, flex-offers are generated frequently (for every intent to use a flexible appliance), and an aggregator agent has to handle all of them efficiently to maintain flexibility model up-to-dateness and inter-level consistency (Requirements 1–2). In this section, we present an incremental version of the N-to-M aggregation approach, allowing to achieve low latency and high throughput while repeatedly aggregating N flex-offers to M aggregated flex-offers.

Consider two pools (sets) of flex-offers, F and A , that are maintained continuously over time such that $A = \text{reduce}^A(F)$. The pool F is updated using a sequence of incoming updates: u_1, u_2, \dots, u_k . Each update u_i is of the form (f, c_i) , where f is a flex-offer and $c_i \in \{+, -\}$ indicates insertion ('+') or deletion ('-') of f to/from F . Our proposed incremental aggregation approach outputs a sequence of updates of aggregated flex-offers in A resulting after applying u_1, u_2, \dots, u_k to F . The approach has four phases: *grouping*, *optimization*, *bin-packing*, and *aggregation*.

Grouping phase. We map each flex-offer into a d -dimensional point. This point belongs to a cell in a d -dimensional uniform grid. Users specify the extent of each cell in each dimension using tolerances T_1, T_2, \dots, T_d which are defined as the part of grouping parameters (see Section 6.4). Every cell is identified by its coordinates in the grid. We only keep track of *populated* cells, using an in-memory hash table, denoted as the *group*

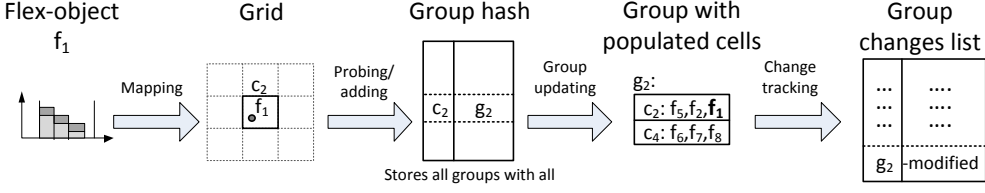


Figure 6.5: The steps of processing an addition of a flex-offer in the grouping phase

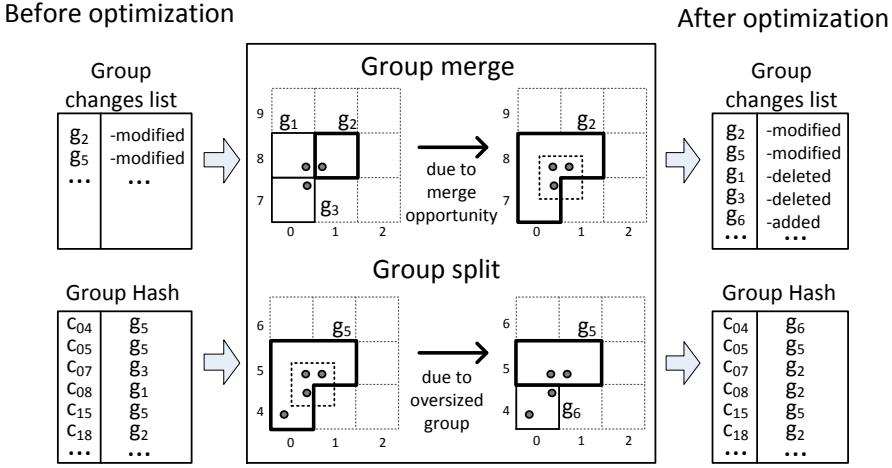


Figure 6.6: The flow of data in the optimization phase

hash. This table stores key-value pairs, where the key is the cell coordinates and the value is the set of flex-offers from F mapped to this cell. We combine adjacent populated cells into a *group*. The group can be either *created*, *deleted*, or *modified*. Group changes are stored in a list, denoted as the *group changes list*. Figure 6.5 visualizes the effect of adding a flex-offer f_1 . First, f_1 is mapped to a 2-dimensional point which lies in the grid cell c_2 . The coordinates of c_2 are used to locate a group in the group hash. The found group is updated by inserting f_1 into a flex-offer list associated to the group. Finally, a change record indicating that the group was *modified* is inserted into the group changes list. In the case when a group is not found in the group hash, a new group with a unique *id* and a single populated cell c_2 is created. Also, if the *group changes list* already contains a change record for a particular group, the record is updated to reflect the combination of the changes.

Optimization phase. This phase begins the “delta aggregation” and is only ex-

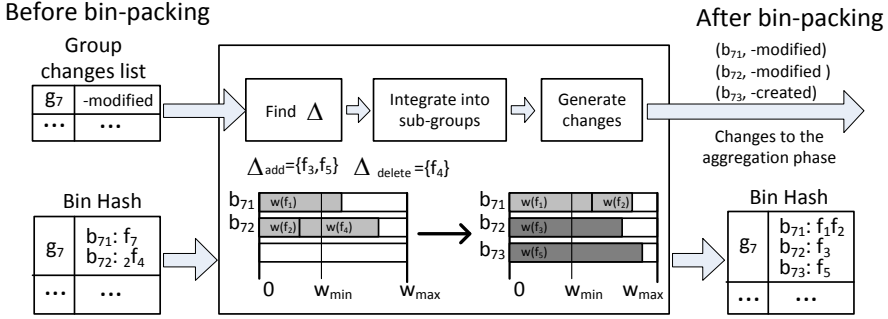


Figure 6.7: The flow of data in the bin packing phase

ecuted when A updates are requested, i.e., either (1) periodically, (2) after a certain number of updates, or (3) when the content of up-to-date A is requested. During this phase, we consolidate the group changes list. For each update of a group g in the list, we identify its adjacent groups g_1^a, g_2^a, \dots by probing the group-hash. Then, for each adjacent group g_i^a , a minimum bounding rectangle (MBR) is computed over all points, which contains flex-offers from the groups g and g_i^a . If the extent of the MBR in all dimensions is within the user-specified tolerances, we combine the groups g and g_i^a (see merge in Figure 6.6). Otherwise, if the MBR of g in any dimension is larger than the size of a grid cell, we perform a group split (depicted in Figure 6.6). Any over-sized group is partitioned into groups of a single grid cell, and, for every individual group, an MBR is computed. Then, the two groups with the closest MBRs are merged until the grouping constraint is violated. Then, g is substituted with newly built groups. Groups changes incurred during merging and splitting are added to the group change list.

Bin-packing phase. We maintain a hash table, denoted the *bin hash*, which maps each group (produced in the grouping phase) to its bins (as described in Section 6.4). In this phase, we propagate updates from the group change list to bins. We first compare existing bins with an updated group to compute the deltas to obtain added and deleted flex-offers, Δ_{added} and Δ_{delete} , respectively. Then, we discard from the bins the flex-offers that are in Δ_{delete} . Groups with the total weight less than w_{min} are deleted and flex-offers from these groups as well as from Δ_{added} are included into other existing bins using the first fit decreasing strategy [50]. New bins are created, if needed. Figure 6.7 shows how the bins of the group g_7 are updated when lower and upper bounds w_{min} and w_{max} are pre-set. Finally, all bins changes are pipelined to the aggregation phase. Flex-offers that did not fit any bin (due to their weight being lower than w_{min} or higher than w_{max}) are stored in a separate list.

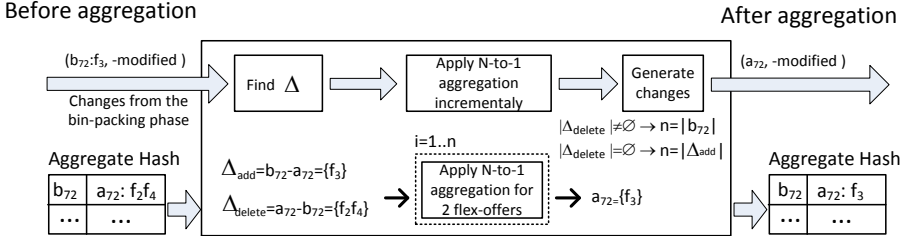


Figure 6.8: The flow of data in the aggregation phase

Aggregation phase. We maintain a hash table, denoted as the *aggregate hash*, which maps each individual bin to an aggregated flex-offer. Each aggregated flex-offer has references to the original flex-offers. Thus, for every bin change, added and deleted non-aggregated flex-offers (see Δ_{add} and Δ_{delete} in Figure 6.8) are found and used to incrementally update an aggregate flex-offer. If there are no deletes, N-to-1 aggregation is incrementally applied for every added flex-offer. Otherwise, an aggregated flex-offer is recomputed from scratch by applying N-to-1 aggregation on all flex-offers in a bin. Finally, all changed aggregated flex-offers together with a change type (*added*, *removed*, *modified*) are provided as output.

6.6 Performance Study

In this section, we study the performance of our proposed incremental N-to-M aggregation approach (*reduce^A*). For the study, we develop two implementations: *stand-alone* and *integrated*. The stand-alone implementation performs aggregation of a given set of flex-offers, as elaborated in Sections 6.2–6.5, and is independent of other PRESCRIPTIVECPS agent components. Additionally, the integrated implementation offers an integrated scheduling of flex-offers (*reduce^B*) and thus allows optimizing the BRP's costs by producing (good enough) flex-offer assignments after they are aggregated with our N-to-M approach. The details and the results of the experiments with these two implementations are elaborated in the two subsequent sections.

6.6.1 Stand-alone aggregation study

We now present the experimental evaluation of the full incremental N-to-M aggregation approach, when used as the stand-alone implementation. As there are no other solutions for flex-offer aggregation and disaggregation, we propose two rival implementations: *Hierarchical Aggregation* and *SimGB*. In *Hierarchical Aggregation*, we use agglomerative

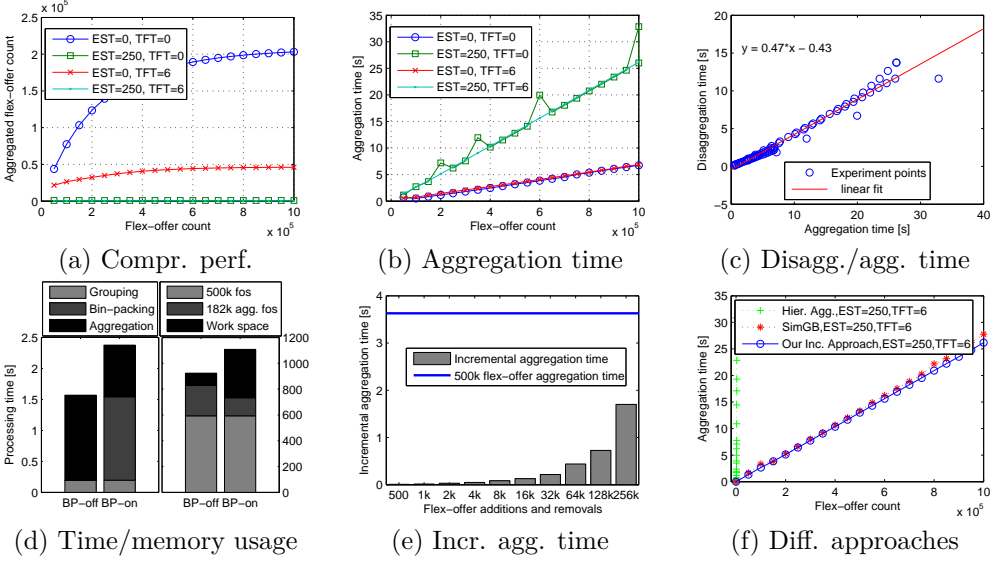


Figure 6.9: The results of the scalability and incremental behaviour evaluation

hierarchical clustering for the grouping phase. First, the approach assigns each flex-offer to individual clusters. Then, while no grouping constraints are violated, it incrementally merges the two closest clusters. The distance between two clusters is calculated based on the values of flex-offer attributes specified in grouping parameters. For *SimGB*, we apply the *similarity group-by operator* [51] for one grouping parameter at a time, thus partitioning the input into valid groups of similar flex-offers.

Experimental setup

For the evaluation, we use a synthetic flex-offer dataset from the MIRABEL project. The dataset contains one million energy consumption flex-offers and the time is discretized at every 15 min. The *earliest start time* (t_{es}) is distributed uniformly in the range $[0, 23228]$. The number of slices and the time flexibility values ($t_{ls} - t_{es}$) follow the normal distributions $\mathcal{N}(8, 4)$ and $\mathcal{N}(20, 10)$ in the ranges $[10, 30]$ and $[4, 12]$, respectively; the slice duration is fixed to 1 time unit for all flex-offers, thus profiles are from 2.5 to 7.5 hours long. Experiments were run on a PC with Quad Core Intel®Xeon®E5320 CPU, 16GB RAM, OpenSUSE 11.4 (x86_64), and Java 1.6. Unless otherwise mentioned, the default values of the experiment parameters are:

- The number of flex-offers is 500k.
- $EST = 0$ (*Earliest Start Time Tolerance*) and $TFT = 0$ (*Time Flexibility Tol-*

erance) are used as the grouping parameters. They apply on the *Earliest Start Time* (t_{es}) and *Time Flexibility* ($t_{ls} - t_{es}$) flex-offer attributes, respectively.

- The aggregate constraint is unset (bin-packing is disabled). We also perform experiments with bin-packing enabled (explicitly stated).

Results and discussion

Results of different experiment categories are presented below.

Scalability. For evaluating flex-offer count reduction (compression) performance and scalability, the number of flex-offers is gradually increased from $50k$ to $1000k$. Aggregation is performed using two different *EST* and *TFT* parameter values: *EST* equal to 0 or 250, and *TFT* equal to 0 or 6. Disaggregation is executed with randomly generated flex-offer assignments of aggregated flex-offers. The results are shown in Figure 6.9(a-d). Figure 6.9(a-b) shows that different aggregation parameter values lead to different flex-offer count reduction (compression) factors and aggregation times. Disaggregation is approx. 2 times faster than aggregation (see Figure 6.9(c)) regardless of the flex-offer count and grouping parameter values. Most of the time is spent in the bin-packing (if enabled) and N-to-1 aggregation phases (the 2 left bars in Figure 6.9(d)). Considering the overhead associated with incremental behaviour, the amount of memory used by the approach is relatively small compared to the footprint of the original and aggregated flex-offers. Memory usage increases when bin-packing is enabled.

Incremental Behaviour. When evaluating incremental aggregation performance, we first aggregate $500k$ flex-offers. Then, for different k values ranging from 500 to $256k$, we insert k new flex-offers and remove k randomly selected flex-offers. The total number of flex-offers stays at $500k$. For every value of k , we execute incremental aggregation. As seen from Figure 6.9(e), the updates can be processed efficiently, so our approach offers substantial time savings compared to the case when all $500k$ flex-offers are aggregated from scratch (the horizontal line in the figure). We then compare the total time to process flex-offers with our incremental approach to the other two (inherently non-incremental) approaches - *Hierarchical Aggregation* and *SimGB*. As seen in Figure 6.9(f), our incremental approach is competitive to SimGB in terms of scalability. The overhead associated with the change tracking in our approach is not significant in the overall aggregation time. Additionally, the hierarchical clustering-based approach (Hier. Agg.) incurs very high processing time even for small datasets (due to a large amount of distance computations) and is thus not scalable enough for this aggregation problem.

Grouping Parameters Effect. As seen in Figure 6.10(a), the *EST* significantly affects the flex-offer count reduction (compression) factor. For this dataset, increasing *EST* by a factor of two leads to a flex-offer reduction by approximately the same factor. However, the use of high *EST* values results in aggregated flex-offer profiles with more slices. Aggregating these requires more time (see “aggregation time” in Figure 6.10(a)). The *TFT* parameter has a significant impact on the flexibility loss (see “flexibility

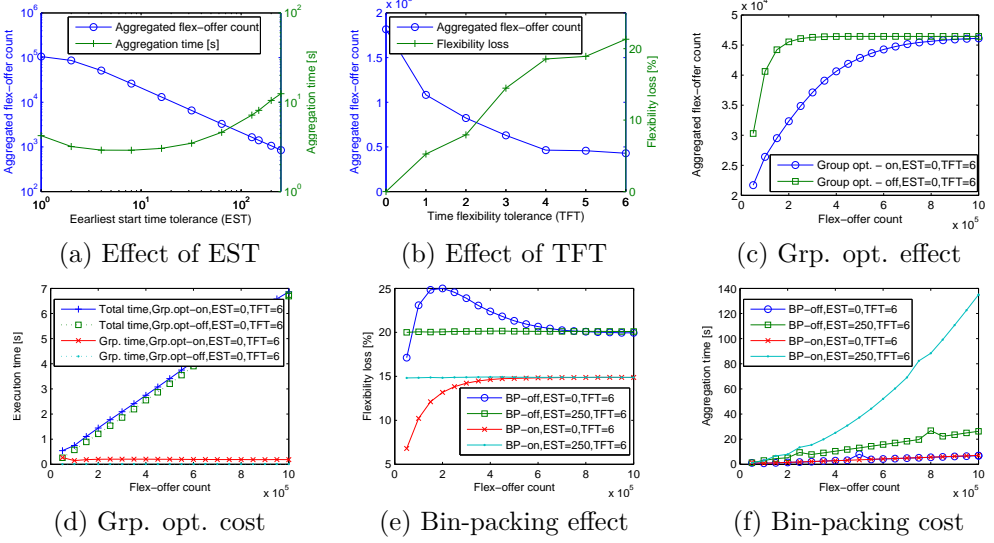


Figure 6.10: The results of the grouping, optimization, and bin-packing evaluation

loss” in Figure 6.10(b)). Higher values of TFT incur higher flexibility losses. When it is set to 0, aggregation incurs no flexibility loss, but results in a larger amount of aggregated flex-offers. When the number of distinct time flexibility values in a flex-offer dataset is low (as in our case), the best compression with no flexibility losses can be achieved when $TFT = 0$ and the other grouping parameters are unset (or set to high values). However, due to the long durations of profiles and high total amount values, the produced aggregated flex-offer might violate the aggregate constraint.

Optimization and Bin-packing. We now study the optimization and bin-packing phases. As seen in Figure 6.10(c-d), the optimization phase is relatively cheap (Figure 6.10(d)), and it substantially contributes to the aggregated flex-offer count reduction (Figure 6.10(c)). For bin-packing evaluation, the aggregate constraint was set so that the time flexibility of an aggregate is always at least 8 ($w_{min} = 8$, equiv. to 2 hours). By enabling this constraint, we investigate the overhead associated to bin-packing and its effect on the flexibility loss. As seen in Figure 6.10(e), by bounding the time flexibility for every aggregate, the overall flexibility loss can be limited. However, bin-packing introduces a substantial overhead that depends on the number of flex-offers in groups after the optimization phase (see Figure 6.10(f)). When this number is small ($EST = 0$, $TFT = 6$), the overhead of bin-packing is insignificant. However, when groups are large ($EST = 250$, $TFT = 6$), bin-packing overhead becomes very significant.

In summary, we show that our incremental aggregation approach scales linearly in the number of flex-offer inserts. The overhead associated with incremental behaviour is

insignificant. Our approach performs aggregation *incrementally* just as fast as efficient non-incremental grouping approaches (*SimGB*). The flex-offer count reduction and flexibility loss can be controlled using the grouping parameters. The count reduction can be further increased efficiently by group optimization. Disaggregation is approx. 2 times faster than aggregation.

6.6.2 Integrated aggregation study

We now present the experimental evaluation of the integrated implementation, where the incremental N-to-M aggregation approach is coupled with flex-offer scheduling for the aim of immediate energy balancing after flex-offers are aggregated. In the MIRABEL use-case, such joint aggregation and scheduling workflow is pursued by aggregator and BRP agents, respectively, as elaborated in Chapter 2, and corresponds to applying *reduce^A* and *reduce^B* operations, elaborated in Section 3.6.

The use-case scenario

For the experiment, we assume the following typical MIRABEL scenario: on the day-ahead market, the BRP buys a certain amount of energy for all 24 hours of the following day and thus commits itself to balance the acquired production with the respective consumption at every hour. If, for a particular hour, the energy bought by the BRP does not match the consumed one, the BRP has to pay a penalty that is calculated based on the amount of imbalance energy and its price. Therefore, one hour before the energy delivery day starts, the BRP agent utilizes aggregated flex-offers (collected from an aggregator agent) to balance energy demand and supply for the subsequent 24 hours with the objective to minimize the total imbalances and thus to maximize its profit. In our experimental setting, we assume that the BRP agent collects flex-offers of consumption only and is isolated from a TSO agent, i.e., there is no exchange of flex-offers and flex-offer assignments with a TSO agent. The maximum scheduling time is fixed to 10 min, leaving at least 50 min for propagation of flex-offer assignments back to prosumer agents (incl., time for disaggregation and communication of all flex-offer assignments).

Experimental setup

We use the implementation presented in Section 6.6.1 and integrate into it three existing flex-offer scheduling algorithms [52]: evolutionary (EA), local optimization (LO), and random search (RS). The scheduling algorithm was executed ten times in order to obtain more reliable estimate. The maximum scheduling time is 10 min, but the algorithm stops earlier if its best result has not been improved for one minute. The scheduling was performed with and without the prior aggregation of flex-offers. When aggregation

was used, three tolerances were used as grouping parameters: *earliest start time tolerance* EST , *time flexibility tolerance*, TFT , and *profile duration tolerance* PDT . These tolerances were set to two extreme values, 0 and ∞ (no bound is set), and to several intermediate values. For the experiment, we use the same machine and configuration, as presented in Section 6.6.1. The properties of the used flex-offer dataset are as follows:

- The dataset contains 100k flex-offers (a subset of the dataset from Section 6.6.1).
- Flex-offer attributes in a dataset follow these distributions and have the following bounds: $t_{es} \sim \mathcal{U}(0, 96)$, $1 \leq t_{es} \leq 92$; $flex_T(f) \sim \mathcal{N}(8, 4)$, $4 \leq flex_T(f) \leq 12$; and $p_{dur}(f) \sim \mathcal{N}(10, 10)$, $1 \leq p_{dur}(f) \leq 20$.
- Profile slice durations are fixed to 1 time unit (15 min), i.e., $\forall s \in P(f) : s.d = 1$.

In other words, flex-offers start between 0:00 and 23:00, their start time flexibility varies from 1 up to 3 hours, and their profile durations vary from 15 min to 5 hours. We assume that the BRP buys an amount of energy equal to that defined by all 100k flex-offers; this energy is distributed following the typical daily energy usage pattern (more energy used in a day time, less at night). Additionally, we use real imbalance and retail energy prices from the Slovenian electricity market.

Results and discussion

Figure 6.11 shows the average imbalance after multiple runs of scheduling were performed with all of our chosen combinations of the aggregation parameters, in addition to, with no aggregation performed (see the marks at 100k flex-offers). As we can see, the flex-offer count has almost no direct influence on the scheduling result. Moreover, the RS performs worse compared to the LO and the EA. The evolution principles of the EA bring an advantage when more than 40 aggregated flex-offers need to be scheduled. When there is no aggregation, the LO does not find a single solution in the given amount of time, while the EA computes only the initial random population, achieving the same results as the RS. This means that some aggregation is needed to produce good results.

The remaining imbalance in the non-aggregated case is compared to the best found imbalance by the EA in Figure 6.12. The combination of aggregation and scheduling successfully minimizes the cost for the BRP (and consequently the remaining imbalance), leaving some imbalance only in the beginning and at the end of the 24-hour interval. More specifically, if the flex-offers are favourably aggregated, the remaining imbalance equals only 5% of the remaining imbalance in the non-aggregated case.

To study the effect of aggregation on the scheduling results, we need to take a closer look at the aggregation parameters. Figure 6.13 presents the individual aggregation parameter impact on the aggregated flex-offer count and the average scheduling results found by the EA. As we can see, aggregation parameters EST , TFT , and PDT contribute similarly to flex-offer count reduction, but they have a different impact on the

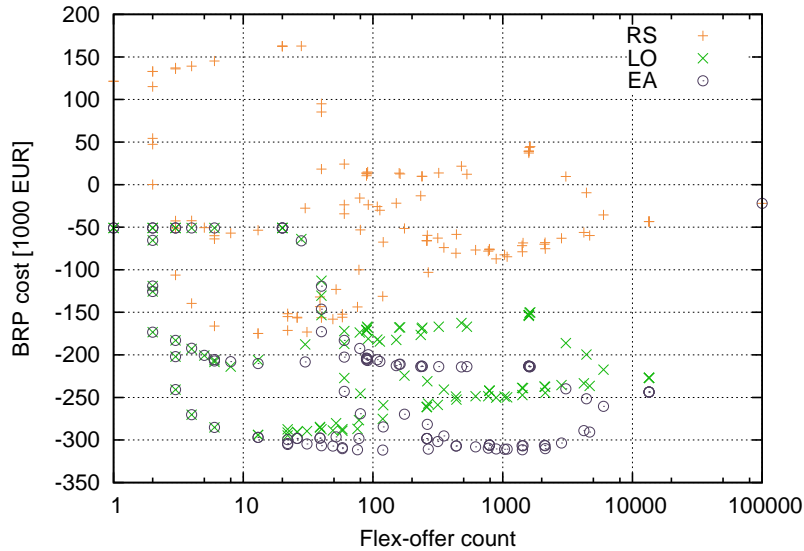


Figure 6.11: The influence of (aggregated) flex-offer count on (average) scheduling result

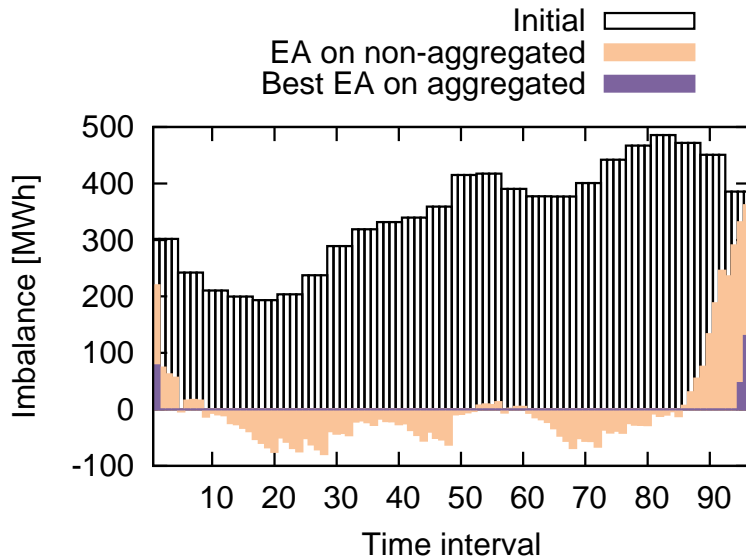


Figure 6.12: Aggregation impact on the remaining imbalance found by the EA

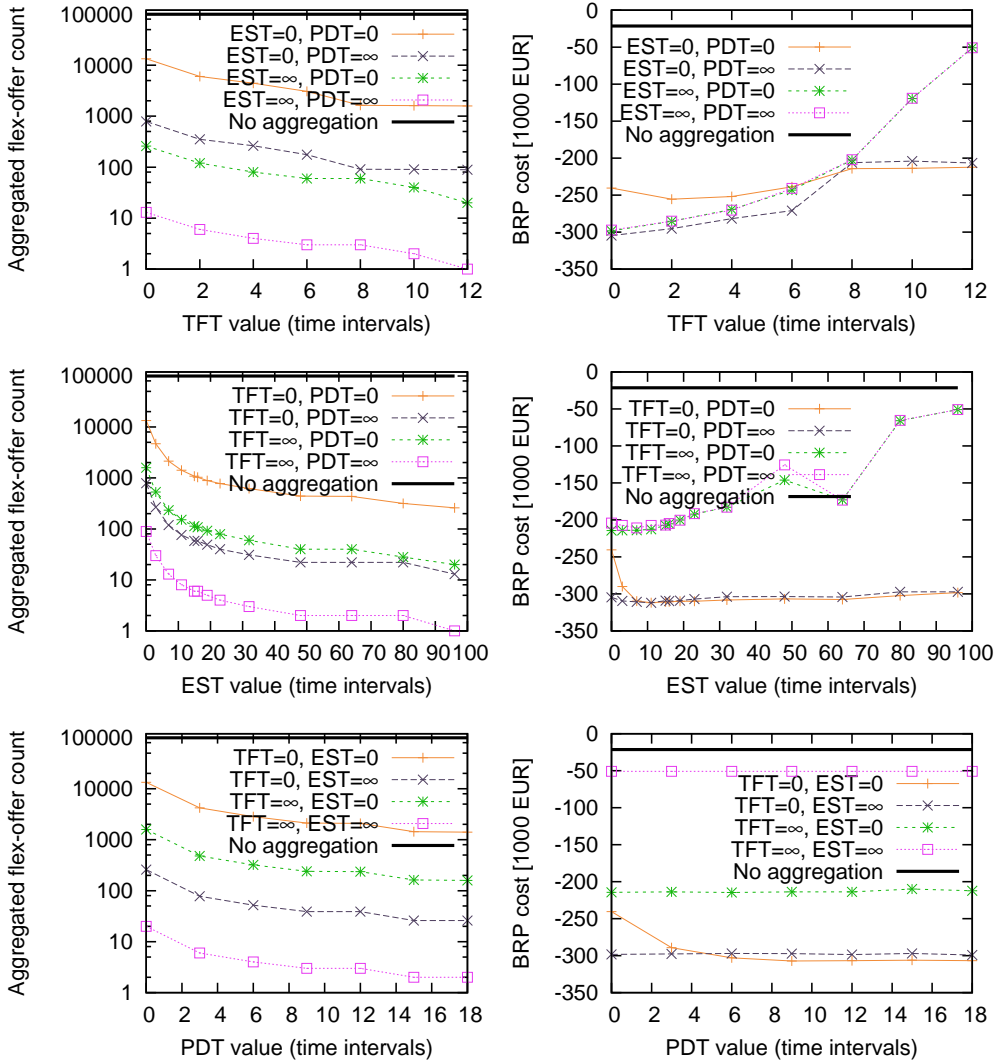


Figure 6.13: Aggregation parameters impact on flex-offer count (left column) and average scheduling result by the EA (right column)

quality of results. Specifically, keeping the value of TFT as low as possible almost always guarantees better scheduling results compared to higher TFT values. This means that in order to obtain better results, aggregation should preserve as much time flexibility ($flex_T$) as possible, which is exactly what low TFT values achieve. The EST parameter has a different impact depending on whether $TFT = 0$ or $TFT = \infty$. When $TFT = \infty$, then long aggregated flex-offer (obtained with high EST values) normally result in worse scheduling results comparing to short aggregated flex-offers (obtained with low EST values). However, when $TFT = 0$, the scheduling result can be improved by lowering EST value until the increased aggregated flex-offer count starts to dominate and thus negatively influence scheduling. For example, the overall best scheduling results were achieved with the EA when $TFT = 0$ and $EST = 7$ or 11 . Finding the best EST value is another (meta-) optimization problem. The third parameter PDT has little impact on the scheduling result. While it decreases the aggregated flex-offer count, this does not contribute achieving better scheduling results.

In summary, the aggregation parameters TFT and EST are the most significant for scheduling. When tuned and set correctly, an aggregation allows improving scheduling results by up to 20 times, compared to the case when no aggregation is performed.

6.7 Related Work

The research related to the flex-offer aggregation problem fall in several categories.

Clustering. Many clustering algorithms have been proposed, including density-based (e.g., BIRCH [53]), centroid-based (e.g., K-Means [54]), hierarchical clustering (e.g., SLINK [55]), and incremental algorithms such as incremental K-means [56] and incremental BIRCH [57]. In comparison to our approach, clustering solves only the grouping part of the problem, which is a lot simpler than the whole problem. For the grouping alone, the closest work is incremental grid-based clustering [58–60], where we, in comparison, improve the clusters across the grid boundaries and limit the number of items per each cluster.

Similarity Group By. SimDB [61] groups tuples in a database based on the similarity between tuple values, and is implemented as a DBMS operator in [51]. However, SimDB again only solves the grouping part of the problem, and is (unlike our approach) not incremental, which is essential for the low-latency high-performance processing of flexibility models (Requirements 1–2).

Complex objects. Complex objects with multidimensional data exist in many real-world applications [62] and can be represented with *multidimensional data models* [63]. Several research efforts (e.g., [64] and [65]) have been proposed to aggregate complex objects. However, these efforts do not consider the specific challenges related to aggregating flex-offers.

Temporal Aggregation. Several papers have addressed aggregation for temporal and spatio-temporal data including: instantaneous temporal aggregation [66], cu-

mulative temporal aggregation [67–69], histogram-based aggregation [70] and multi-dimensional temporal aggregation [71]. These techniques differ in the way how a time line is partitioned into time intervals and how an aggregation group is associated with each time instant. The efficient computation of these time intervals poses a great challenge and therefore various techniques that allow computing them efficiently are proposed [72–74]. Unfortunately, these techniques only deal with simple data items without flexibilities, making them unsuitable for aggregation of flex-offers.

6.8 Summary and Discussion

We focused on the lossy *reduce*^A operation, which aggregates set of flex-offers (multiple *unit* flexibility models) into a set of aggregated flex-offers (a *multi-unit* flexibility model), and it is used by an aggregator agent in the MIRABEL use-case. We formally defined relevant concepts, including measures for stimuli and responses, and provided a novel and efficient grid-based *reduce*^A approach while considering the grouping of flex-offers, alternatives for computing aggregates, the disaggregation (*map*^A) process, and the correctness requirement associated to these. The incremental variant of the aggregation approach was presented. It allows achieving low latency and high throughput when flex-offers are aggregated repeatedly. Extensive experiments with a stand-alone *reduce*^A implementation and data from the MIRABEL project showed that the approach provided reasonable performance and is suitable for the use in the MIRABEL CPS setting. Experiments based on the MIRABEL use-case and while using the prototype with integrated aggregation and scheduling (*reduce*^A and *reduce*^B) showed that the aggregation allows improving scheduling result up to 20 times (leaving up to 5 % of the imbalance of the non-aggregated case) when scheduling time is limited and aggregation parameters are set correctly. Here, *TFT* and *EST* are two most significant aggregation parameters, allowing to group flex-offers with similar starting times and time flexibilities and resulting into aggregated flex-offers with much time flexibility preserved.

Other similar instances of PRESCRIPTIVECPS may also use similar lossy flexibility model aggregation approaches, during which flexibility models (e.g., *unit* model) are firstly grouped based on their similarity, then additionally (re-)grouped to satisfy some *aggregate constraint* applicable to each group (or an aggregate), and finally, for each group, merged into the components of some “larger” flexibility model (e.g., *multi-unit* model). Aggregation techniques presented in this chapter may support only the grouping, re-grouping (bin-packing), but not the merging phase of the aggregation. For the merging phase, additional merging (N-to-1 aggregation) functions must be specified for the specific types of flexibility models used as inputs and outputs of *reduce*. An incremental aggregation should be considered (1) to increase aggregation performance and/or (2) to minimize communication between an aggregator agent and an adjacent agent consuming output from *reduce*.

Chapter 7

Flexibility Model Forecasting

In this chapter, we consider PRESCRIPTIVECPS agents in the *forecasting* role and focus on the *build* operation allowing to build flexibility models based on context information and/or historical measurements collected from other agents or external sources (see Section 3.5). The chapter is split into two parts. The first part (Section 7.1) elaborates on how flexibility models with invariant responses (see Section 3.2) defined as time series could be forecasted using traditional domain-independent forecasting methods (e.g., *regression* or *exponential smoothing*) inside a DBMS – used as part of the agent’s software system (Chapter 4). The second part (Section 7.2) presents domain-dependent techniques for converting (augmenting) forecasted invariant flexibility models into variant flexibility models with lots of stimuli and responses. In both parts, we use the MIRABEL use-case example to demonstrate the proposed concepts.

7.1 Invariant Flexibility Model Forecasting in DBMS

In this section, we consider the design of a forecasting-capable DBMS used as part of the agent’s software system (Chapter 4) and show how a standard DBMS technology can be extended to support the forecasting of traditional *time series* with one *time* and one or multiple *measurement* attributes. As demonstrated in Section 3.6, such time series is a special type of a flexibility model with an invariant response (e.g., $ts^{cons+prod}$) and may be used as a common building element of various more complex flexibility models (e.g., the *BG flexibility model*). Our proposed DBMS design is based on the traditional model-based time series forecasting process which, when integrated into a DBMS, allows for additional optimizations and advanced DBMS functionalities that improve the efficiency, consistency, and transparency of the overall forecasting process. The proposed DBMS extensions are generic and can support invariant flexibility model forecasting for the MIRABEL’s as well as in other instances of PRESCRIPTIVECPS (Requirement 7).

We first discuss the MIRABEL’s use-case of forecasting, summarize the requirements entailed by this use-case, and shortly discuss related work for each of these requirements (Section 7.1.2). We then present a general architecture of a forecasting DBMS allowing to address the defined requirements and explain how such forecasting DBMS can be used in the MIRABEL use-case to forecast energy consumption (Section 7.1.3). Finally, we give a brief overview over remaining research topics and challenges in Section 7.1.5. The content of this section is based on Publication [5].

7.1.1 Built-in forecasting in MIRABEL

In the MIRABEL use-case, a BRP agent takes the *forecasting* role, in which it uses the *build^B* operation to forecast inflexible consumption and production loads in the form of time series ($ts^{cons+prod}$). For forecasting, it uses historical energy consumption and production as well as weather measurements collected from external data sources (e.g., DSO, meteorological stations). Accurate and timely forecasts are needed by the BRP agent to “complete” instances of the BG flexibility model used for balancing demand and supply and thus ensuring the stability and energy efficiency within a balance group (see Section 3.6). To realize the *build^B* operation and to meet the requirements for historical measurements storage (Requirement 8), measurement-based forecasting (Requirement 7), efficiency (Requirement 1), and correctness (Requirement 2), we propose the use of in-DBMS forecasting of time series.

Forecasting time series ($ts^{cons+prod}$) in MIRABEL can be seen as an instance of the traditional model-based time series forecasting process (see Figure 7.1) that consists of three main phases – *model creation* (identification and estimation), *model usage* (forecasting), and *model maintenance* (evaluation and adaption). First, the model creation phase involves selecting and building a stochastic model that captures the dependency of future on past data. This is an expensive process as parameter estimation of many sophisticated models involves numerical optimization methods that iterate several times over the base data. However, once a model is created and parameters are estimated, it can cheaply be used over and over again to forecast future values of the time series (model usage). The model maintenance phase evaluates new actual data of the time series and triggers possible adaption of the forecast model. This is computationally expensive as well, as most parameters cannot be maintained incrementally and thus, again parameter estimation is required.

The tight coupling of the described forecasting process within a DBMS (1) ensures consistency between data and models, (2) increases efficiency by reducing data transfer and exploiting database related optimization techniques, and (3) enables declarative forecast queries (see Section 4.3) for any user. In contrast to partial integration approaches that reuse statistical tools like R within the DBMS (e.g., Ricardo [75]), we argue for a full integration approach that might require a higher initial effort but allows for optimizations on the forecasting process itself.

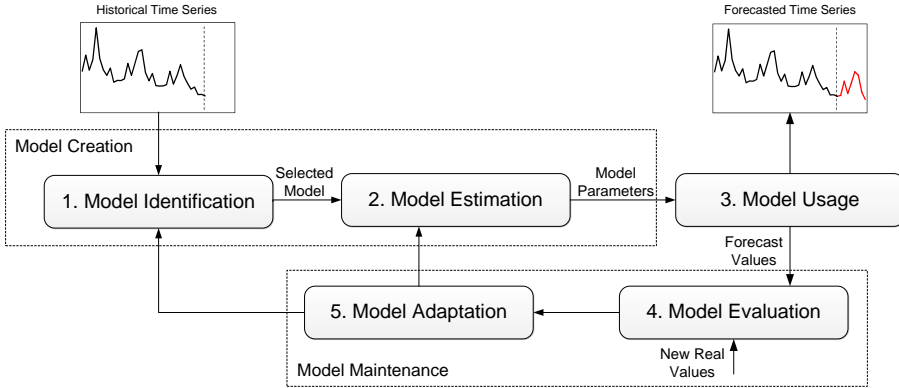


Figure 7.1: The general process of forecasting

7.1.2 Requirements and related work

Following the MIRABEL use-case, we now elaborate the general requirements for forecasting (Requirement 7), correctness (Requirement 2), and efficiency (Requirement 1) using a number of more detailed requirements specific for forecasting in a database. Additionally, we overview related work for each of these detailed requirements.

Advanced Forecasting Functionality First of all, the database system should provide advanced statistical forecasting functionalities that provide high accuracy for various use-cases and time series data. For example, in the energy domain (the MIRABEL use-case) multi-equation forecast models are often necessary to achieve reasonable accuracy [76]. Also, new forecasting methods required by applications should be easily addable. Major commercial DBMSs support only a limited amount of such forecasting functionality. For example, Oracle offers linear as well as non-linear regression methods or exponential smoothing as part of its OLAP DML [77]. The Data Mining Extension (DMX) in Microsoft SQL Server supports a hybrid forecast method consisting of ARIMA and autoregressive trees [78].

Declarative Forecast Queries Forecast queries (see Section 4.3) should follow the traditional SQL interface and offer a simple language extension usable for any user. For example, Duan et. al. [79] proposed a simple FORECAST keyword to specify declarative forecast queries.

Integrated into Relational Query Processing Forecast queries should be seamlessly integrated into standard relational query processing allowing arbitrary forecast

queries as well queries on forecasted query results (e.g., joins of forecasted and historical data). Within commercial DBMSs, predictive functionality is usually implemented as customized functions using proprietary languages [77, 78] and thus, cannot be utilized with other relational operators. In contrast, Paris et. al. [80] developed a formal definition of a forecast operator and explored the integration of forecast operators with standard relational operators.

Transparent and Automatic Query Processing The processing of forecast queries should be done transparently and automatically by the database system. This includes automatic creation or reuse of forecast models for given forecast queries as well as automatic maintenance of materialized forecast models. For example, within the Fa system [79] an incremental approach was proposed to automatically build ad-hoc models for multi-dimensional time series, where more attributes are added to the model in successive iterations.

Efficient Query Processing Complex forecast models or large amount of time series data might lead to long execution time of forecast queries. Thus, optimization techniques are required that efficiently process such forecast queries. Ge and Zdonik [81] proposed an I/O-conscious skip list data structure for very large time series. In addition, techniques to efficiently reuse models exploiting multi-dimensional data have been developed [82–84].

Efficient Update Processing Finally, a continuous stream of new time series values requires efficient maintenance of computed models. General [85] and model-specific [86] techniques to determine when a model requires maintenance have been proposed. In addition, existing approaches speed up the computation process itself by parallelization [87] or by using previous model parameters [88].

To summarize, existing research papers already identified and addressed individual aspects of the requirements in the area of time series forecasting. However, no general architecture addressing all these requirements and exploiting existing optimization approaches has been described so far. In the next section, we present a general architecture that addresses this issue.

7.1.3 The architecture of a DBMS for forecasting

We now give a high level overview of the conceptual architecture of forecasting DBMS and then elaborate extensions that are specific to a built-in forecasting functionality.

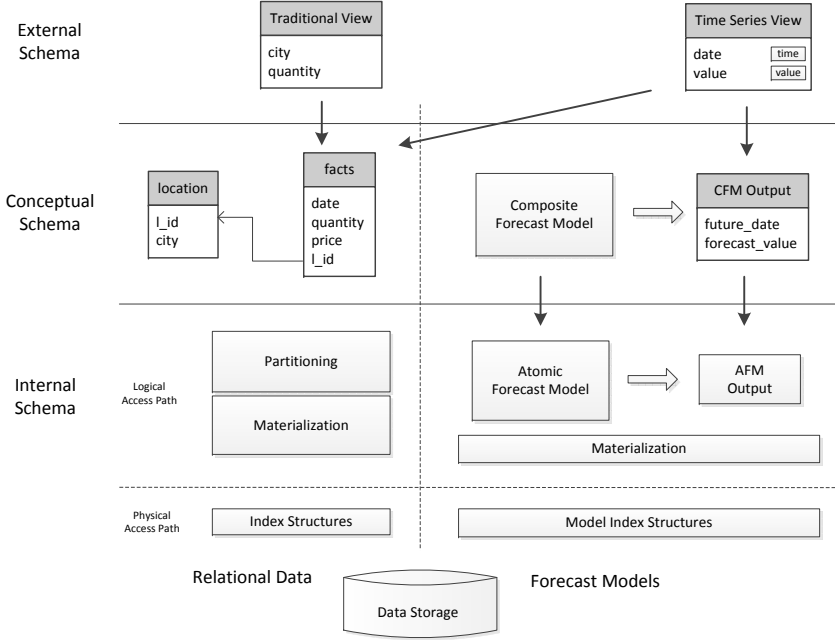


Figure 7.2: The extension of the 3-layer ANSI/SPARC architecture

Overview

The basic idea underlying our forecasting DBMS approach is to develop a DBMS architecture that specifically supports and is optimized for the execution of *forecast queries*, i.e., queries that involve forecasted values. For this purpose, we decided to base our approach on the standard ANSI/SPARC architecture [89] and enhance it with specific forecasting components. In particular, we propose specific changes and additions on all three levels of the ANSI/SPARC architecture to allow a transparent and efficient end-to-end execution of forecast queries. Figure 7.2 illustrates the ANSI/SPARC architecture as well as our proposed additions. Our changes on the external schema level mainly comprise the definition of a special *Time Series View* that ensures a representation of data in combination to a time axis. On the conceptual schema level, we define a *Composite Forecast Model* that is a conceptual representation of a concrete (atomic) forecast model. Composite models can define a forecast model composition, meaning that the forecast model is decomposed into multiple individual forecast models. With this approach, we can, on the one hand, achieve a higher accuracy by intelligently modelling compositions. On the other hand, we are able to reduce maintenance costs by reusing models in multiple compositions. The internal schema is divided into the Logical

Access Path [90] and the Physical Access Path. The Logical Access Path comprises of the *Atomic Forecast Model* that is a concrete realization of the forecast model defining the forecast model type and characteristics. Further on this level, we propose a possible materialization of forecast models and forecasting results to quickly provide results for frequently repeating forecast queries. On the Physical Access Path, we suggest using special *Forecast Model Index Structures* that further increase the efficiency when answering forecast queries [83]. Additionally, specific time series data index structures [81] might be introduced to speed up time series access. In the following, we describe our enhancements for the ANSI/SPARC architecture in more detail and show the application of our proposed concepts using the MIRABEL use-case.

System architecture details

External Schema The external schema in the ANSI/SPARC architecture comprises user-defined data views, which can be seen as virtual tables storing the results of specific queries. A view can comprise attributes of multiple tables as well as pre-defined aggregations or calculation results. Forecasts are typically calculated on time series data, meaning a sequence of discrete values measured successively over time. To allow forecast queries in a database system, we define a special type of a regular view that ensures the representation of data as time series. Thus, the *Time Series View* comprises of an obligatory time attribute containing discrete points in time in its natural order and one or multiple attributes exhibiting measurements at these specified moments. Optionally, these attributes can be tagged with forecasting-specific meta data, which, for example, indicates whether an attribute represents a dependent variable to be forecasted or an independent variable such as *external influence* having an impact on the main variable. Typically, there is one main variable and many external influences. An example query defining a *Time Series View* is denoted as:

```
CREATE TIMESERIESVIEW tv1 AS
SELECT date AS TIME, energy AS MAIN_VAR, temperature AS INFLUENCE
FROM measurements
ORDER BY TIME;
```

A time series view can represent both historical and forecasted values of time series. Typically, after its creation only historic values are contained, but as soon as a query requests future values, these values are forecasted and added to the view accordingly. For predicted values, the view also contains further information such as the standard deviation or confidence intervals, which clearly distinguishes future values from historic values. Once real values are available, they replace the forecasted values. The time series view is typically defined by a user or an application, and, once defined, it is automatically managed by the DBMS (e.g., by automatically calculating forecasts or adding new values). The view can be queried in an ad-hoc fashion at any time and can be referenced by any other regular view or time series view. In some cases, as we

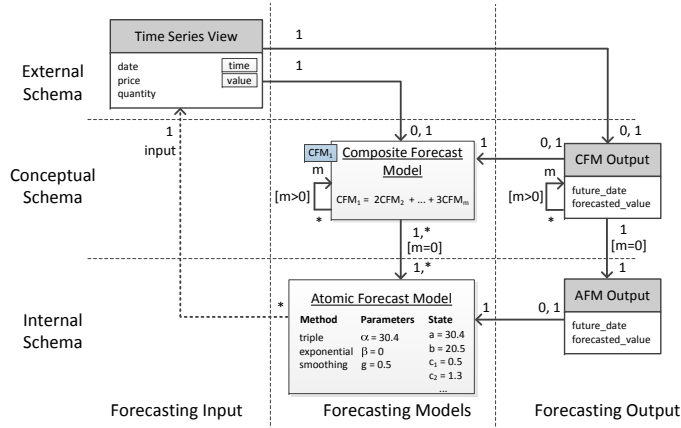


Figure 7.3: Relationship between forecasting components

explain later, a time series view is generated automatically by the DBMS as the result of forecast model decomposition.

Conceptual Schema The conceptual organization of the data in an ANSI/SPARC compliant DBMS is defined in the conceptual schema level. This level comprises of a data schema that describes available entities, their relationship and contained attributes and can be seen as an abstraction from the logical and physical data representation. Likewise, we define *Composite Forecast Models* as a conceptual abstraction from concrete (atomic) forecast models and thus, it can be seen as some kind of transparency layer. As illustrated in Figure 7.3, each composite forecast model in the hierarchy can either reference multiple child composite forecast models or, on the leaf level, ultimately refer to atomic forecast models defined in the internal schema layer of the ANSI/SPARC architecture. Examples of simple and more complex conceptual forecast models are given below.

Example 7.1 (Conceptual forecast model of a solar panel production).

In a simple case, a conceptual model directly refers to a single atomic forecast model from the internal schema, representing a simple direct forecast, e.g., energy production of a single solar panel.

Example 7.2 (Conceptual forecast model of production in Germany). *In a more complex case, composite forecast models can also describe a hierarchical forecast composition. When forecasting the energy consumption of Germany, for example, the*

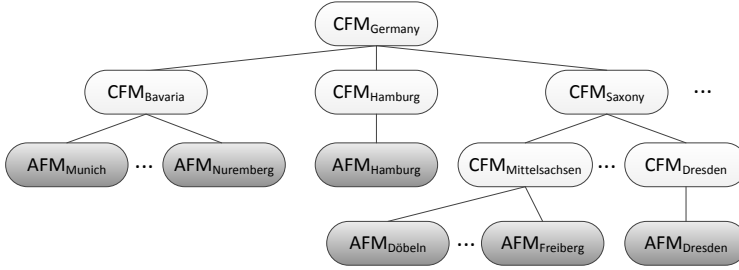


Figure 7.4: The hierarchy of forecast models for energy consumption in Germany

forecasting can be decomposed into forecasts of the energy consumption for all German states, or further down in the hierarchy, the energy consumption of all German districts and cities, as shown in Figure 7.4. In such case, the composite forecast model can define a hierarchical forecast composition referring further composite forecast models on multiple hierarchy levels. The final forecast is later calculated by aggregating the single results of the referenced atomic forecast models according to the defined hierarchical composition.

It is important to note that the automatic determination of forecasting compositions is a complex task with many prerequisites and constraints. For now, we assume that compositions are pre-defined by the database administrator that is aware of the database schema and the available data. We discuss the automatic composition creation separately in Section 7.1.4.

Besides the definition of forecasting compositions, typically, one composite forecast model references only one atomic forecast model (see Figure 7.3). However, for the sake of forecasting accuracy it is also possible to employ ensemble forecasting [91,92], where multiple atomic forecast models of different types and with different parameter combinations are executed in parallel. Afterwards, the results are combined in a weighted linear combination, where the most accurate forecast model gains the highest weight. In this case, a single composite forecast model might refer to multiple atomic forecast models.

With respect to the external layer, each composite forecast model is linked with a single time series view from the external schema. It further defines a single output (“CFM Output”), which is a special table complying to the same rules as the introduced time series view.

Internal Schema The logical and the physical data access paths are defined in the internal schema of an ANSI/SPARC architecture. Logical access paths refer to aspects like partitioning and materialization while the physical access paths define low level access structures like indexes. Likewise, we define *Atomic Forecast Models* that represent a non-decomposable forecast model. A single atomic forecast model is represented by 1) an *input*, 2) the *forecast model type*, 3) *forecast model parameters* and 4) the current *forecast state* (see Figure 7.3). Here, the input is the data as defined in the associated time series view, referenced through the connected composite forecast model. The forecast model type (e.g., exponential smoothing or ARIMA) is chosen from a *forecast model catalog* that represents all forecast model types available in the DBMS and is pre-defined with respect to the application domain and the common data characteristics. The chosen forecast model type determines the forecast model characteristics (e.g., number of lags) and defines the parameters of the forecast model. When creating an instance of an atomic forecast model the parameters are estimated using local (e.g., LBFGS) or global (e.g., simulated annealing) general purpose optimization algorithms. This estimation involves a large number of simulations and thus, typically is very time consuming. We show how such optimization algorithms can be integrated into a DBMS in Chapter 8. Finally, the output of the atomic forecast model—predicted future values—is stored in a special data structure called the *atomic forecast model output* (“AFM Output”). This table comprises of at least a time column and exactly one value column that contains the forecasted values. Optionally, additional attributes might be included in the atomic forecast model output (e.g., prediction intervals).

Similarly to materialized views, composite and atomic forecast models can be computed on the fly or materialized for faster query response times. Materialized forecast models store the forecast model parameters and the forecast model state. This especially avoids the very time consuming estimation of the forecast model parameters and thus, allows a very fast provisioning of forecasting results. As a result, the execution time of forecast queries is greatly reduced. A further reduction is possible when directly materializing the forecasting results, i.e., the forecast model output. Similarly to materialized views, materialized forecast models require maintenance after each appended time series value. This includes either the simple update of the forecast (when the model is still accurate) or a more expensive parameter re-estimation (when the model violates the accuracy constraints). The maintenance of the materialized forecast models can be conducted asynchronously to the execution of forecast queries, which allows for fast forecast query execution at all times. For the physical access paths of the internal schema, we define model index structures that allow an efficient storage and search for materialized composite forecast models and connected atomic forecast models [83]. Such index structures ensure efficient updates (or invalidations) of atomic forecast models on time series updates as well as efficient access to instantiated atomic forecast models and their outputs during the execution of forecast queries. We also enhance the classical data index structures to allow efficient processing of time series data. First, we employ

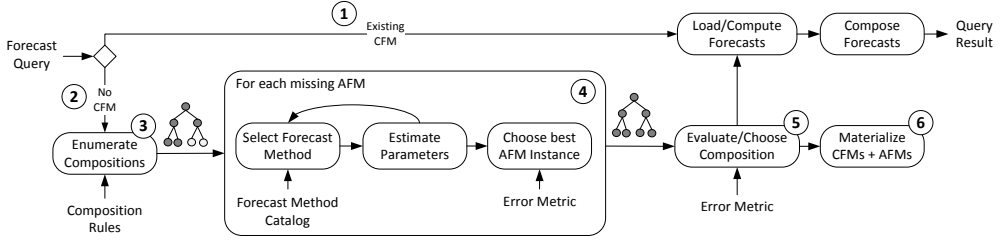


Figure 7.5: The steps of processing forecast queries

time index structures to ensure that the time series values can be accessed in a timely subsequent order as it is required for forecast models. Second, it is also possible to employ more advanced indexing structures like skip-lists [81] or similarity indexes, which would further increase the processing efficiency and decrease the forecast query response times.

7.1.4 Processing forecast queries

Based on the introduced conceptual architecture, we now discuss the processing of a forecast query. We first give a high level overview of the basic steps in forecast query processing (Section 7.1.4) and then traverse the process in more detail using an energy forecasting example from the MIRABEL use-case (Section 7.1.4).

Overview

We distinguish two main cases to process a given forecast query (Figure 7.5). First, if a suitable composite forecast model exists, we compute or load (either model parameters or materialized forecast values) the forecasts for each atomic forecast model within the given composite model and compose the final forecasts according to the stored composition rule ((1) in Figure 7.5). Second, if no composite forecast model is available, we have two choices. We can either return an error to the user or we can compute a composite forecast model on the fly ((2) in Figure 7.5). In the latter case, we first enumerate different composition alternatives using a *composition rule catalog* or composition advisor (3). Such a catalog might store meta data describing the hierarchical dependencies in the data that can be used as composition rules. If no such rule catalog is available, we do not create a forecasting composition, but create a single composite forecast model that attaches exactly one atomic forecast model to the given time series view. For all found composition alternatives, we then create all missing atomic forecast models that do not already exist in the database (4). Such an atomic forecast model is created by empirically evaluating different forecast methods that are stored in the *forecast method catalog* and choosing the best one or by employing forecast model ensembles. Finally,

Table 7.1: An example of the energy consumption relation (“measurements”)

Date	Group	Amount
2012-01-01 09:00	1 (households)	200
2012-01-01 09:00	2 (small industries)	500
2012-01-01 09:00	3 (large industries)	1500
2012-01-01 09:15	1	250
2012-01-01 09:15	2	525
2012-01-01 09:15	3	1600
...

after all atomic forecast models have been created, we choose the composition approach with the lowest error and calculate the query result (5). The previously described forecasting process, including the model composition creation, is conducted transparently to the user. The user simply requests a forecast for the user-defined time series view, and the system automatically decides upon the necessary steps to provide the results. Finally, to avoid expensive parameter estimation for future forecast queries, the system might choose to materialize the final composite and corresponding atomic forecast models, including the model parameters and model state (6). In the following, we further detail this process of automatic composite forecast model creation.

The example of energy forecasting in MIRABEL

We now show how our enhanced DBMS architecture can be utilized in the MIRABEL use-case to forecast the total inflexible consumption within a balance group ($t^{cons} \in t^{cons+prod}$). Suppose that BRP’s collected data includes aggregated power measurements from households, small, and large industrial consumers, consuming electricity within the balance group, and such energy consumption data of different consumer groups (households, small/large industries) is stored like shown in Table 7.1. Initially, a *time series view* over this data, aggregating energy consumption measurements, is created:

```
CREATE TIMESERIESVIEW tsBGconsTotal AS
SELECT Date AS TIME, SUM(Amount) AS MAIN_VAR AS Amount
FROM measurements
WHERE Group BETWEEN 1 AND 3
GROUP BY TIME ORDER BY TIME
```

Now, suppose a user submits the following query over this time series view:

```
SELECT TIME, Amount
FROM tsBGconsTotal
WHERE TIME IN (yesterday(), tomorrow())
```

The system seamlessly determines if the query involves forecasting (accesses future values) or not. As the user query in our example requests the total energy consumption for the next day, the system automatically triggers a search for a corresponding *composite forecast model* in the DBMS. In our example, no corresponding model is found, and the system seamlessly creates a new composite forecast model. The system has many different alternatives to create this model, and it might spend some time on choosing an alternative, which offers the best forecasting accuracy. Suppose the composition rule catalog outputs a very simple composition rule that suggests aggregating the forecasts of individual consumer groups (1, 2, and 3) to retrieve the overall energy consumption forecast. Further assume that two composite forecast models, CM_2 and CM_3 , already exist in the database (as they are used by other time series views previously defined by a user) to forecast the consumption of small and large industries. To evaluate this composition rule, the system needs to create an additional composite forecast model for energy consumption of private households CM_1 . This model CM_1 requires an input, defined by the following automatically generated time series query:

```
SELECT Date AS TIME, Amount AS MAIN_VAR
WHERE Group = 1 (i.e., households)
FROM measurements
GROUP BY TIME ORDER BY TIME
```

When creating CM_1 , the system creates underlying atomic forecast model AM_1 and empirically evaluates the forecast methods listed in the *forecast method catalogue*. For AM_1 , in our example, the forecast method *Triple Seasonal Exponential Smoothing* [93] is chosen as the most accurate solution for forecasting household consumption data for a specific data set. Then, the models CM_1 , CM_2 , and CM_3 are composed using the following rule $\alpha \cdot CM_1 + \beta \cdot CM_2 + \gamma \cdot CM_3$. α , β and γ are the weights of the linear combination describing the impact of the respective consumer groups. Typically, the weights reflect the share of each consumer group on the total consumption, which is computed from the history of the corresponding time series.

The accuracy of this composition rule might be compared to other composition rules (e.g., create only one composite model for the overall energy consumption time series `tsBGconsTotal`), which also require the creation of missing atomic forecast models. Finally the composition rule producing the most accurate forecasts is chosen (the aggregation of individual groups in our example) and the output to the query is obtained by aggregating the forecast values from the outputs of CM_1 , CM_2 , and CM_3 (see “LFM Output” in Figure 7.2) at respective time stamps. Then, the output is merged with historical values of `tsBGconsTotal` from yesterday and the result set is returned to the user.

Finally, the system might choose to materialize the composite and corresponding atomic forecast models including, for our example, CM_1 and the parameters of AM_1 , to speed up the processing of future queries. Additionally, the forecasting result might be materialized.

7.1.5 Research topics & challenges

Realizing a forecast-enabled DBMS, which takes advantage of our proposed architecture, is challenging. Some aspects were already addressed by individual research papers, as discussed in Section 7.1.2. In this section, we focus on remaining challenges that either nobody has addressed yet or that arise due to our novel architecture.

External Schema. Queries of different types have to be supported, processed, and optimized to take advantage of the models stored within a DBMS. These include traditional ad-hoc as well as reoccurring and continuous forecast queries. Existing SQL extensions [94] might be further refined to allow seamless querying of time series data that does not require the specification of a `FORECAST` keyword, e.g., by restricting the time in the where clause of a query (**WHERE TIME IN** (`now()`, `tomorrow()`)). Now the system additionally needs to detect if a certain time series query involves forecasting or just demands the history of the time series. In addition, query constraints might be specified on the desired accuracy or runtime. This requires anytime or online approaches that progressively provide better forecast results over time.

Conceptual Schema. Each forecast query on time series view requires either the use of an existing composite forecast model or a new model needs to be built. Large databases might contain millions of time series [82], requiring efficient strategies to search and build composite forecast models. The decomposition of composite forecast models into a hierarchy of multiple composite forecast models can be either done manually or automatically. Automatic approaches face two main challenges. First, they need to determine what decompositions are possible and, second, they need to choose the best decomposition. Suitable decompositions might be given by the database administrator in terms of composition rules, derived automatically from the data (e.g., by using foreign-key relationships) or given by meta data like data cubes or data hierarchy information. The determination of the most accurate decomposition is quite a hard problem as the number of possible decomposition might be very high and as the accuracy of decomposition cannot be determined without actually building all concerning models [95]. First approaches in this area [84,96] are only suitable for a small number of time series. Additionally, the system might transparently maintain composite forecast models. Thus, the system can automatically switch to a new composition if it leads to higher forecast accuracy.

Internal Schema—Logical Access Path. As forecast queries should be processed transparently, atomic models have to be chosen and created automatically. Due to the large variety of possible models and parameterizations, this process of model identification is challenging. Domain-specific model types can reduce the search space by including only models from a given domain. Automatization approaches are required

that automatically select the “best” model for a time series. First approaches automate the model selection process for ARIMA [97] or Exponential Smoothing [98] models. Through the usage of ensemble models, i.e., combination of several individual atomic models, the robustness and accuracy of such approaches might be improved. Once one or several models have been selected, model parameters need to be estimated, requiring an optimization problem to be solved. Chapter 8 of this thesis concerns formulation and in-DBMS solving of such optimization problems.

To avoid expensive model creation, atomic forecast models might be materialized so they can be used over and over again. However, due to evolving time series in many domains, materialized models require maintenance in form of parameter reestimation. Research in this area focuses on two main challenges, first, when to reestimate parameters and, second, how to speed up the reestimation itself. Existing research papers already address both challenges (as discussed in Section 7.1.2) and might be extended and improved.

Due to expensive model maintenance, materialized atomic forecast models have to be selected carefully. The question arises how to intelligently reuse models in order to keep maintenance costs as low as possible while enabling high accuracy (e.g., one atomic forecast model might be used in several composite forecast models). Such a configuration of atomic forecast models might be chosen offline by a system administrator or online using automatic approaches, which, in addition to model maintenance, requires continuous evaluation and adaption.

Internal Schema—Physical Access Path. Finally, specific index structures on time series data or forecast models might be used to speed up model creation, usage and maintenance – as discussed in Section 7.1.3. These initial approaches might be improved by more advanced index structures, either for a specific model type or the general case. In addition, partitioning the data with index structures might additionally enable parallelization approaches that parallelize within or between parameter estimators and models. One such parallelization approach was proposed for an energy-domain-specific forecast model [87] and might be extended to other model types.

To sum up, multiple research aspects remain on all three layers of the ANSI/SPARC architecture and open up many interesting research directions. In contrast to traditional query processing, all forecasting relevant topics (e.g., selection, estimation, maintenance) need to cope with a two dimensional optimization objective - forecast accuracy versus runtime of forecast query processing.

7.1.6 Summary

In this section, we presented the design of a DBMS offering declarative, transparent, and efficient *forecast queries*, which allow projecting historical data into future and thus

building instances of flexibility models (Requirement 7). A DBMS relying on our proposed design can offer many optimization opportunities, which are not possible when using the traditional (non-integrated) model-based time series forecasting process. Following the MIRABEL use-case, we first described a forecasting-capable DBMS using a set of forecasting-specific requirements and, for each of these requirements, reviewed related work. Then, we introduced the generic architecture of a forecasting DBMS allowing to address the defined requirements. The architecture is based on the traditional ANSI/SPARC architecture and includes a number of forecasting components at different abstraction layers. We also explained how such forecasting DBMS can be used in the MIRABEL use-case to forecast energy consumption. Finally, we briefly overviewed remaining research topics and challenges.

Forecasting of traditional time series, as presented in this section, allows creating simple flexibility models with *invariant responses* only (see Section 3.2), and it is sufficient for the MIRABEL use-case, where inflexible consumption and production loads ($ts^{cons+prod}$) need to be forecasted. Other instances of PRESCRIPTIVECPS, in general, may require forecasting of more complex flexibility models with lots of stimulus and response variants, which need to be derived (forecasted) based on historical data. A simple approach to forecast such (variant) flexibility models would decompose model instances into a number of time series and apply (in-DBMS) forecasting for each of these time series individually to produce a desired (variant) instance of a forecasted model. However, various (complex) constraints and/or dependencies over model attributes might not be satisfied or captured using such approach. Consequently, in the next section, we follow a different approach and demonstrate five domain-specific techniques, allowing to convert (augment) invariant flexibility models to variant flexibility models.

7.2 Variant Flexibility Model Forecasting

In this section, we demonstrate how invariant flexibility models (like forecasted *time series*) could be converted (augmented) into variant flexibility models defining lots of stimulus and response variants. We demonstrate this conversion in the context of MIRABEL when generating instances of the *unit* flexibility model, which is represented as a (single) flex-offer and used by agents in the *prosumer* role (see Section 3.6). For the conversion, we present five domain-specific techniques allowing to generate flex-offers based on forecasted household's (prosumer's) energy consumption measurements, represented as time series (an invariant flexibility model). The content of this section is based on Publication [6].

7.2.1 Generic flex-offer generation architecture

Suppose that the objective is to generate one or more flex-offers (i.e., instances of the *unit* flexibility model) based on historical energy consumption in a single household. For this,

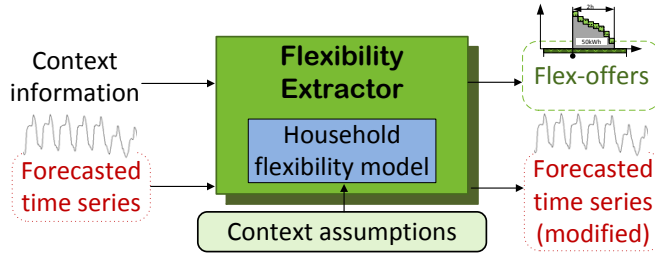


Figure 7.6: The generic flex-offer generation architecture

we propose a two-step procedure. In the first step, we build a time series by forecasting energy consumption in the household (e.g., using the technique from Section 7.1). In the second step, we extract energy amounts (responses) and associated flexibilities for changing these amounts (stimuli) an energy consumer affords in its (daily) energy consumption in the household, and represent such amounts and flexibilities using flex-offers. We now focus on the second step of this procedure and elaborate how such extraction can be done.

In the general case, energy amounts and flexibilities of a household highly depend on the types of appliances consuming energy and on individual energy consumer using these appliances. For the realistic generation of flex-offers, the detailed model of individual household (and a consumer) is required. We propose a general architecture for a flex-offer generation, shown in Figure 7.6. Given a *forecasted time series* and *context information* as input, the so-called *flexibility extractor* decomposes (disaggregates) the given time series into flexible and inflexible parts, assuming that not all energy is flexible. These parts are returned as flex-offers and a (modified) time series in the output, respectively. According to the provided context information, the extractor internally builds and uses an instance of a *household flexibility model*, incorporating (context) assumptions about the household’s energy flexibility patterns.

7.2.2 Flex-offer generation approaches

As shown in Figure 7.7, we propose two general classes of flexibility extraction approaches that are compatible with the general architecture (shown in Figure 7.6). The first class includes approaches that operate at the level of total (aggregated) household consumption and output flex-offers for the complete household. In contrast, the second class includes approaches that operate at an individual appliance level and output flex-offers for each flexible use of an appliance (e.g., dishwasher). We now elaborate (some of the many possible) flexibility generation approaches in each of these classes.

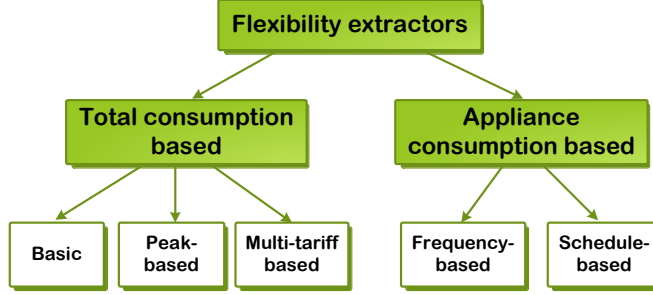


Figure 7.7: Proposed flex-offer generation approaches

Basic approach assumes that some percentage of the household consumption is flexible, and this flexibility is available at any time of the day. The percentage of the flexible energy part is specified as the *context information*, together with other flex-offer-related information, e.g., the number of *intervals* in a flex-offer, *interval duration*, *minimum* and *maximum percentage* of required energy, *creation time*, *acceptance time*, *assignment time*, *earliest start time*, and *latest start time* (see Chapter 5 for more).

Peak-based approach assumes that an energy in a household is only flexible at peaks of energy consumption (e.g., at 7-8 PM when a consumer is back from work). The number of peaks within a day and the durations of all peaks are determined during the analysis of time series. Energy extracted at peak intervals are represented as flex-offers, where the percentage of the flexible energy part during the peak interval, together with other information, is specified as the *context information*, like for the basic approach.

Multi-tariff approach assumes that information about each household's energy flexibility is known in advance and, in this case, is calculated by comparing household energy consumption patterns before and after multi-tariff (or hour-tariff) billing system is introduced to energy consumers. When a multi-tariff is introduced to the consumer, it is then expected that the consumer consumes more during low-tariff periods and less during high-tariff periods. Such household-specific consumption flexibility patterns are then used to generate flex-offers accordingly.

Frequency-based approach generates appliance-level flex-offers based on pre-specified appliance consumption profiles (e.g., obtained from manufacturers) and *frequencies* of the expected (daily or weekly) use of appliances, all specified as *context information*. The approach detects activations of appliances (i.e., use of energy from appliances) based on a provided frequencies and time series and then generates flex-offers for each such detected activation.

Schedule-based approach generates appliance-level flex-offers based on pre-specified appliance consumption profiles (like in the *frequency-based* approach) and detailed *schedule*

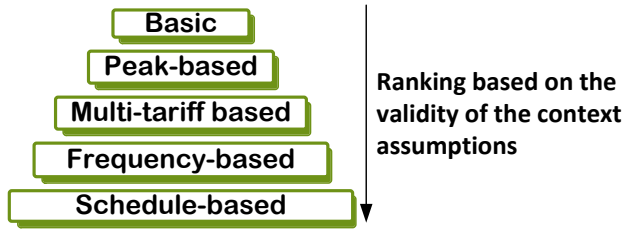


Figure 7.8: Ranking of flex-offer generation approaches

ules of an appliance activation. For each consumer (or groups of consumers), a schedule specifies when a particular appliance (e.g., a dishwasher) is expected to consume energy in a household. Provided such schedules as *context information*, the *schedule-based* approach generates flex-offers for each appliance activation that are detected utilizing the schedule.

The described flexibility generation approaches differ in terms of how realistic context assumptions are (see Figure 7.8). Here, the basic approach is the easiest to implement, requires little context information, but employs least realistic context assumptions leading to less accurate generated flex-offers. On the contrary, the schedule-based approach is the most difficult to implement, requires detail appliance-level context information, but generates flex-offers accurately describing complex household flexibility patterns.

7.2.3 Summary

In this section, we presented a general architecture, and demonstrated five domain-specific approaches, for generating household-level flex-offers based on a time series of forecasted energy consumption in the household. In the context of PRESCRIPTIVECPS, this generation can be framed as the generation (forecasting) of a variant flexibility model instance based on a given (forecasted) invariant flexibility model instance. During the generation, stimuli and variant responses are introduced to flexibility model instances according to a single response alternative (forecasted time series). The forecasting of flexibility models in other PRESCRIPTIVECPS domains can be done based on invariant model augmentation, like it was demonstrated in this section.

7.3 Summary and Discussion

In this chapter, we showed how an agent in the *forecasting* role can, potentially, forecast (generate) flexibility models based on historical measurements (Requirement 7). We presented (1) one generic in-DBMS approach to forecast instances of (invariant) flexibility models defining a single response variant as a tradition time series, and (2) five MIRABEL-specific model-based approaches for forecasting flex-offers – more advanced instances of the *unit flexibility model* with lots of stimulus and response variants. All the presented approaches offer (and demonstrate) an important *predictive analytics* (Section 1.3) functionality required for many other instances of PRESCRIPTIVECPS. Here, the in-DBMS approach relies on a forecasting-capable integrated DBMS. In comparison to the traditional model-based time series forecasting process, this DBMS offers many optimization opportunities and advantages (to users) such as increased efficiency, increased consistency, and declarative forecast queries (Section 4.3). However, the remaining challenges still need to be addressed. When flexibility models with complex stimuli and variant responses need to be forecasted, (invariant) model instances can be decomposed into a number of time series (to be forecasted using the in-DBMS approach) or processed with domain-specific forecasting approaches to produce desired instances of a (variant) flexibility model. For accurate forecasting, the domain-specific approaches have to incorporate detailed context information.

In the context of PRESCRIPTIVECPS, the capability of flexibility model forecasting is important when instances of a flexibility model are required for decision making (*solve*, *solve_M* or *solve_R*) but are not collected from sensor agents (as opposed to flex-offers). In such cases, forecasting is then used to infer stimuli and responses of various physical systems (e.g., inflexible demand and supply). Although different use-cases rely on different flexibility and forecasting models, forecasting can be realized using approaches that are similar to those presented in this chapter. Irrespective to a specific use-case, similar optimization opportunities and challenges may arise.

The next chapter focuses on built-in optimization problem solving. As the forecasting often involves optimization problem solving (e.g., in the parameter estimation step), the functionality and/or components (e.g., solvers) of the DBMS from the next chapter could be greatly reused.

Chapter 8

Decision Model Solving in a DBMS

In this chapter, we consider a PRESCRIPTIVECPS agent in one of the decision making (*global prescription*, *aggregator*, or *disaggregator*) roles and focus on the optimization problem solving-capable DBMS component used as part of the agent's software system (see Chapter 4). First, we argue that the traditional approach to formulate and solve decision (optimization) problems, which rely on data from a database (like in the PRESCRIPTIVECPS case) requires combining an optimization software package with a DBMS - leading to a workflow that is cumbersome, complex, inefficient, and error-prone. Then, we show how a standard DBMS technology can be extended to support decision model formulation and in-DBMS decision model solving (Requirement 10) by presenting a so-called *SolveDB* - a DBMS system which seamlessly integrates data management and optimization problem solving and allows for much simpler and more effective solutions of database-based optimization problems. SolveDB is based on the 3-level ANSI/SPARC architecture and allows formulating, solving, and analysing solutions of optimization problems using a single so-called *solve query*. SolveDB provides (1) an SQL-based syntax for optimization problems, (2) an extensible infrastructure for integrating SolveDB-compliant solvers (each for a specific class of problems, e.g., linear programming), and (3) query optimization techniques to achieve the best execution performance and/or result quality. Extensive experiments with the PostgreSQL-based implementation show that SolveDB offers both much higher tool usability/developer productivity and better overall performance for specification-complex and data-intensive problems. The chapter is structured as follows. Section 8.2 defines a *solve query* and the associated challenges. Section 8.3 presents a *relational solve operator* and its workflow allowing to integrate various solvers. Section 8.4 defines the syntax of solve queries, while Sections 8.5–8.6 describes how solve queries are processed and optimized. Section 8.7 presents the SolveDB

architecture, while Section 8.8 describes the experimental evaluation. Section 8.9 discusses related work and Section 8.10 points to future research directions and discusses the use of SolveDB in the context of PRESCRIPTIVECPS. The content of this chapter is based on Publication [7].

8.1 Built-in Optimization Problem Solving

In the context of PRESCRIPTIVECPS, decision models are built by decision-making agents (in the *global prescription*, *aggregator*, or *disaggregator* roles) based on flexibility models and prescriptions from successor agents (if any) and, when solved, result into prescription models (Section 3.2). The steps of decision model building and solving are carried by the *solve*, *solve^M*, and *solve^R* operations (Section 3.2). Decision models depend much on flexibility and prescription models as well as on business objectives pursued by a particular agent. In the MIRABEL use-case, a decision model used by a BRP agent defines the problem of minimizing imbalances (e.g., via BRP's profit maximization). In the general case, however, a decision model may be based on an arbitrary flexibility model stored in a database (Requirement 4) and define an arbitrary optimization problem, which an agent must be able to solve (Requirement 10).

As a running example of a basic optimization (decision) problem, consider the well-known Sudoku puzzle, where a puzzle setter provides a partially filled 9×9 matrix, and the goal is to complete filling the matrix with the digits 1 to 9 so that each column, each row, and each of the nine 3×3 sub-matrices have no duplicated digits (see Figure 8.1). The initial Sudoku matrix can be represented by the relation shown in Table 8.1a, where the NULL values of the *val* attribute represent Sudoku's unknown digits (decision variables). Then, the Sudoku problem can be viewed as replacing these NULL values with concrete values so that all Sudoku constraints are respected, as shown in Table 8.1b. We call these relations with NULLs (Table 8.1a) and concrete values (Table 8.1b) the *input relation* and *output relation*, respectively. These input and output relations can be seen as the representations of primitive flexibility and prescription models, respectively, used by a PRESCRIPTIVECPS (toy) instance, specialized for the Sudoku problem solving.

To solve the Sudoku optimization problem using existing optimization software (e.g., AMPL or IBM CPLEX), the following complex and I/O intensive workflow is required:

1. The *optimization model* (aka. the decision model) of the Sudoku problem must be created in a modelling language (e.g., OPL, AMPL) with a syntax based on mathematical notation, requiring carefully exploring, simplifying, and abstracting the problem. The Sudoku model can be defined as the (linear) constraint satisfaction problem shown in Figure 8.2 with 729 binary unknown (decision) variables (x), $|G|$ known variables (given elements in G), at least 243 constraint equalities (depending on $|G|$), and a single objective function (the value of which is irrelevant in this case).

<u>2</u>	9	6	4	<u>8</u>	3	<u>7</u>	1	5
8	4	7	5	<u>1</u>	2	3	<u>9</u>	6
1	<u>3</u>	5	<u>6</u>	7	<u>9</u>	4	<u>2</u>	8
6	7	8	3	9	<u>1</u>	<u>5</u>	4	2
<u>4</u>	1	2	8	6	5	9	7	<u>3</u>
3	5	<u>9</u>	<u>7</u>	2	4	8	6	1
9	<u>6</u>	3	<u>2</u>	5	7	1	<u>8</u>	4
5	<u>2</u>	1	9	<u>4</u>	8	6	3	7
7	8	<u>4</u>	1	<u>3</u>	6	2	5	<u>9</u>

Figure 8.1: A Sudoku matrix with given (bold and underlined) and solution (other) digits

Table 8.1: Sudoku input (a) and output (b) relations in decimal format (81 tuples each)

row	col	val
1	1	2
1	2	NULL
...
1	9	NULL
...
9	9	9

(a) input relation (*in_d*)

row	col	val
1	1	2
1	2	5
...
1	9	4
...
9	9	9

(b) output relation (*out_d*)

Table 8.2: Sudoku input (a) and output (b) relations in binary format (729 tuples each)

row	col	val	giv	sel
1	1	1	0	NULL
1	1	2	1	NULL
...
1	1	9	0	NULL
...
9	9	9	1	NULL

(a) input relation (*in_b*)

row	col	val	giv	sel
1	1	1	0	0
1	1	2	1	1
...
1	1	9	0	0
...
9	9	9	1	1

(b) output relation (*out_b*)

$$\begin{aligned}
&\text{Minimize:} && \mathbf{0}^T \mathbf{x} \quad (\text{values are irrelevant}) \\
&\text{Subject To:} \\
&\sum_{j=1:9} x_{i,j,k} = 1, && k, i = 1 : 9 \text{ (only one } k \text{ in each row)} \\
&\sum_{i=1:9} x_{i,j,k} = 1, && k, j = 1 : 9 \text{ (only one } k \text{ in each column)} \\
&\sum_{\substack{i = 3l + 1 : 3l + 3 \\ j = 3m + 1 : 3m + 3}} x_{i,j,k} = 1, && k = 1 : 9, \quad l, m = 0 : 2 \\
&&& \text{(only one } k \text{ in each sub-matrix } M_{l,m}) \\
&x_{ijk} = 1, && \forall (i, j, k) \in G \text{ (given elements } G \text{ are set to "on")} \\
&x_{ijk} = \begin{cases} 1, & \text{if the matrix element } (i, j) \text{ is } k \\ 0, & \text{otherwise} \end{cases}
\end{aligned}$$

Figure 8.2: The mathematical optimization model of the Sudoku 9×9 puzzle

2. *Data bindings* (SQL-based) linking the model elements with database relations must be provided. As the Sudoku model (see Figure 8.2) relies on binary variables (0/1 values) but the relations shown in Table 8.1(a-b) uses decimal variables (values 1–9), additional transformations are included in the data bindings, transforming the input relation in Table 8.1a to the input relation in Table 8.2a, and the output relation in Table 8.2b to the output relation in Table 8.1b.
3. The optimization software processes the model and data bindings, *reading* input relation data, *invoking* a solver, *obtaining* a solution from the solver, and *writing* the solution back to the database output relation.
4. The solution in the output relation may be *validated* and *recomputed*, if needed.

This workflow exhibits a number of *complexities* and *inefficiencies*. First, users must have significant *expertise* in both a database query language (e.g., SQL) and an optimization modelling language (e.g., AMPL), dealing with different types of fundamental structures: relations, tuples, and attribute values in databases versus parameters, variables, constraints, and objectives in optimization models. Second, as data is shipped back and forth between the two systems, significant *performance overhead* is inevitable. Third, users must *manually perform* multiple error-prone actions (create a model and queries, edit data, and execute the model) with different types of software. This is cumbersome and ineffective, particularly, when used as part of a larger decision-making (PRESCRIPTIVECPS) workflow, resulting in frequent changes of the data (and optimization model). Finally, as data, model, and solution are stored in separate and hard-coded

locations, it becomes difficult to *version* models and data so that a particular model version can be used with a particular version of the data.

To address all these problems, we present *SolveDB* - a DBMS that allows performing all these activities using a single so-called *solve query*. The Sudoku problem above can be defined and solved using the following intuitive solve query:

```

1 SELECT out_b.row, out_b.col, out_b.val FROM (
2   SOLVESELECT sel IN
3     (SELECT col, row, v AS val, (val=v) as giv, NULL::boolean AS sel
4       FROM in_d, generate_series(1,9) AS v) AS in_b
5   SUBJECTTO
6     (SELECT sum(sel)=1 FROM in_b GROUP BY val, row),
7     (SELECT sum(sel)=1 FROM in_b GROUP BY val, col),
8     (SELECT sum(sel)=1 FROM in_b GROUP BY val, (col-1) / 3, (row-1) / 3),
9     (SELECT sel = giv FROM in_b WHERE giv),
10    (SELECT sum(sel)=1 FROM in_b GROUP BY row, col)
11   WITH solverlp.cplex() AS out_b
12 WHERE out_b.sel

```

The solve query defines the discussed transformations of the input relation from decimal (*in_d*) to binary (*in_b*) format (lines 3-5) and the output relation from binary (*out_b*) to decimal format (lines 1, 14). Here, *generate_series* is a library function generating the series 1, 2, ..., 9. The constraints of the binary Sudoku problem (see Figure 8.2) are defined as sub-selects in the SOLVESELECT clause (lines 7-12). To resolve these constraints, the query specifies the use of a SolveDB-compliant solver (*solverlp.cplex*) for linear programming (LP) problems.

In general, the novel SOLVESELECT clause produces an output relation from an input relation according to the problem formulations in the inner SUBJECTTO block and an additional inner MINIMIZE/MAXIMIZE clause defining objective functions (not used in the Sudoku query). The SOLVESELECT clause supports multiple attributes with unknown variables of various types (e.g., integers, doubles, or composite types) allowing to represent any imaginable solution as an output relation. To process problem formulations, SolveDB relies on an extensible infrastructure of *solvers* dealing with optimization problems formulated as *views* over an input relation (as in the example above) and other relations in the database. Each new solver registered in SolveDB is either defined in a declarative manner using solve queries based on other existing solvers or developed as a (low-level built-in) function tackling a specific class of optimization problems.

In this chapter, we address a number of challenges related with solve query enabling, formulation, processing, and optimization (not to be confused with the optimization problem solving) and the design of the SolveDB DBMS. To enable solve queries, we present the two-part multi-level workflow to map input and output relations to and from the formats required by various solvers. To formulate solve queries, we define the complete syntax of the SOLVESELECT clause and propose a view-based approach

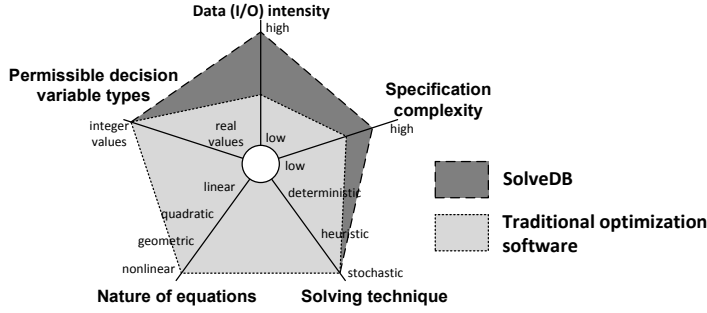


Figure 8.3: SolveDB-targeted types of optimization problems

to formulate optimization problems in a convenient and intuitive way. To process solve queries, we propose an overall SolveDB workflow and the workflow of a typical SolveDB-compliant solver to process SOLVESELECT. We show that solve queries often lead to complex workflows that are expensive to process. Therefore, we propose techniques to tune (optimize) the workflow with the respect to the best execution performance or result quality. Then, based on the introduced concepts, we present the ANSI/SPARC-based architecture of SolveDB. Finally, we perform an experimental evaluation of the PostgreSQL-based implementation of SolveDB with proof-of-concept solvers for LP, black-box, and domain-specific optimization problems, which, compared to traditional tools, demonstrates significantly (1) better tool usability/developer productivity for problems that are complex to specify and (2) increased overall solving performance for problems that are intensive in I/O. Figure 8.3 shows the types of optimization problems which SolveDB is aimed at.

8.2 Solve Queries

We now define the concept of a *solve query* and the associated challenges we will address. As for PRESCRIPTIVECPS (flexibility and prescription) models, we assume the common notion of a problem model, where the *model* is a generic entity (structure) that must be instantiated to a *model instance* [28] before obtaining a problem solution. Following this notion, a *solve query* is a database query that, as a part of its evaluation in SolveDB, invokes a *solver*, defined as follows:

Definition 8.1. Let S be a so-called solver of some optimization problem class. Given the pair (d, p) , the solver produces the solution s in some format (e.g., as a sequence of values of unknown variables) by building and processing (solving) the

model instance of an optimization problem such that $s = S(d, p)$. Here, d is a so-called model instance descriptor (descriptor for short) representing all information needed to build the model instance, and p are configuration parameters (e.g., parameter-value pairs) used for tuning the model instance building and solving process.

Essentially, the solver (S) is capable of “completing the model” by building a model instance (from the supplied d) and solving it with some solving methodology. This general description encompasses most existing types of solvers specialized either for specific or broad optimization problems. At one extreme, S might be specialized for a specific instance of an optimization problem, thus expecting an empty d (less interesting). At the other extreme, S might employ a special format of d (e.g., AMPL) completely defining both the model and the model instance (e.g., the Sudoku model in Figure 8.2 and G). Alternatively, S might be specialized for optimization problems with interchangeable data, where all required data is represented in d (e.g., the built-in Sudoku model from Figure 8.2 with G in d). Although SolveDB is targeted for specification-complex and data-intensive problems (complex and large d), it supports solvers of any of problem classes (e.g., linear programming) exhibiting various properties with regards to the nature of equations, solving technique, permissible variable types, and other aspects (see Figure 8.3). The conceptual and practical integration of solvers into a DBMS leads to a number of interesting challenges:

Challenge 8.1. *How to enable solve queries – map input and output relations to and from descriptors and solutions (d and s) in solver-specific formats?*

Challenge 8.2. *How to formulate solve queries – define descriptors using high-level SQL-based constructs (the desired common language for queries and models) that are intuitive and user-friendly?*

Challenge 8.3. *How to process solve queries – handle descriptors, invoke solvers, and utilize solutions in larger database workflows?*

Challenge 8.4. *How to optimize solve query execution – tune solve query processing to achieve the best execution performance or result quality?*

Challenge 8.5. *How to integrate solve query concepts into existing DBMSs, supporting the full variety of solvers for different optimization problem classes?*

We now show how SolveDB addresses these challenges.

8.3 SolveDB Translation Workflow

To enable solve queries (Challenge 8.1), SolveDB relies on a two-part multi-level (≥ 3 levels) translation workflow to provide a link between user-specific input and output relations and solver-specific descriptors and solutions. We term these two parts the *adaptation workflow* and *abstraction workflow*, respectively. Initially, we focus on the adaptation workflow as it is mandatory for every solve query. Then, we explain how building the abstraction workflow based on the adaptation workflow can further abstract the details and reduce the complexity of problem formulations (and solutions). We conclude the section by defining a *relational solve operator*.

8.3.1 The three-level adaptation workflow

The three-level adaptation workflow produces an output relation denoted as R_{out} from an input relation denoted as R_{in} based on a sequence of views (V) additionally supplied together with R_{in} . The complete adaptation workflow is shown in Figure 8.4 and involves three types of solvers, namely a *physical solver*, a *relational solver*, and a *view solver*.

The *physical solver* (e.g., *cplex*) supports the broadest class of optimization problems, consumes a (physical) descriptor d_p and produces a (physical) solution s_p ; d_p and s_p are represented in internal/native formats, e.g., MPS/SOL.

The *relational solver* translates a descriptor d_r (relational representation) to d_p (internal format), and s_p (internal format) to the solution s_r (relational representation). Here, d_r is a sequence of relations, R_1, \dots, R_N , where each relation (potentially) represents different parts of the model and employs different solver-specific schemas. s_r is a binary relation $R_s(var_nr, value)$ representing the numbers (var_nr) and the found values ($value$) of unknown variables. Depending on if d_r completely represents all information in d_i , the relational solver may offer all or only some of the physical solver's capabilities.

The *view solver* uses (all or some of) the capabilities of the relational solver to solve the problem instance defined by the descriptor d_v to produce a solution s_v , both d_v and s_v in a so-called *view-based* (*view* for short) representation. Here, s_v is an output relation R_{out} which is a special view over R_{in} as exemplified earlier. The descriptor d_v is a 3-tuple (R_{in}, U, V) , where R_{in} is an input relation, U is the set of attribute names representing unknown variables (e.g., $\{val\}$ in Table 8.1a or $\{sel\}$ in Table 8.2a), and $V = V_1, V_2, \dots, V_L$ is the sequence of view definitions (e.g., queries in SQL) over the relation R_{in} and, potentially, other relations in the database. To make the translation of d_v possible, these views as well as the attributes in U must comply with the requirements specific to a particular view solver, e.g., setting a limit on the number and the data types of the U attributes, and requiring that each view definition $V_i, i=1..L$, represents a relation in some solver-specific schema. We later elaborate on how such views are defined.

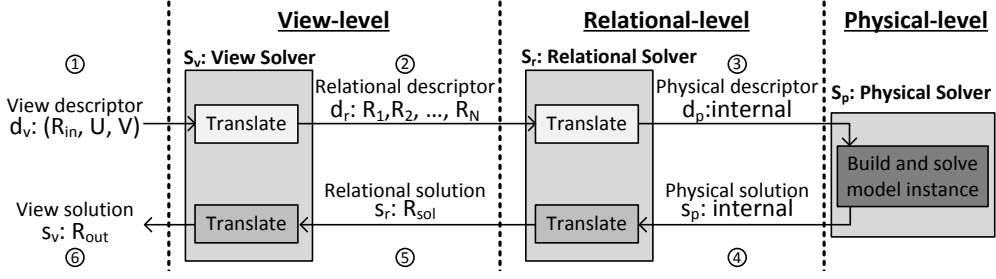


Figure 8.4: The three-level adaptation workflow to build and solve a model instance defined by d_v

Example 8.1 (Sudoku query adaptation workflow). *SolveDB* invokes such an adaptation workflow when processing the `SOLVESELECT` clause (lines 2-13) from the Sudoku solve query in Section 8.1. First, a descriptor $d_v = (R_{in}, \{sel\}, V_1, V_2, \dots, V_5)$ is generated. Here, R_{in} is the binary Sudoku relation (`in_b`) from Table 8.2a and V_1, V_2, \dots, V_5 are the five select statements from the `SUBJECTTO` block. Later, d_v is consumed by the view solver `solverlp` which builds the relational descriptor d_r based on R_{in}, U , and V such that $d_r = R_1, R_2, \dots, R_5$. The relational solver uses d_r to produce d_i in the internal matrix-based format (e.g., MPS) and then, by using d_i as input, invokes the physical solver `cplex` for LP problems. Finally, if a solution is found, it is propagated backwards through all the solvers, leading to the binary Sudoku output relation (`out_b`) from Table 8.2b. This process is further elaborated in Section 8.5.2.

To summarize, view solvers offer a relational view-based approach to supply descriptors to and obtain solutions from various physical solvers, leading to the three-level adaptation workflow. However, descriptors and solutions may still be too detailed and too hard to deal with for users having insufficient understanding of how to define cumbersome constraints (and objectives) supported by the underlying physical solver (consider all 243 Sudoku constraints in the format of the LP solver). Thus, SolveDB also offers an additional problem description and solution abstraction layer, leading to the multi-level abstraction workflow, described next.

8.3.2 The multi-level abstraction workflow

To provide a problem formulation that is convenient for a user and not just for a solver, so-called *composite* view solvers are used by SolveDB. As an example, consider the *composite* Sudoku view solver (`sudoku_composite_solver`) used in the following solve query:

```
SOLVESELECT val IN (SELECT col,row,val FROM in_d) AS r
SUBJECTTO (SELECT val!=4 FROM r WHERE row=2 AND col=4)
```

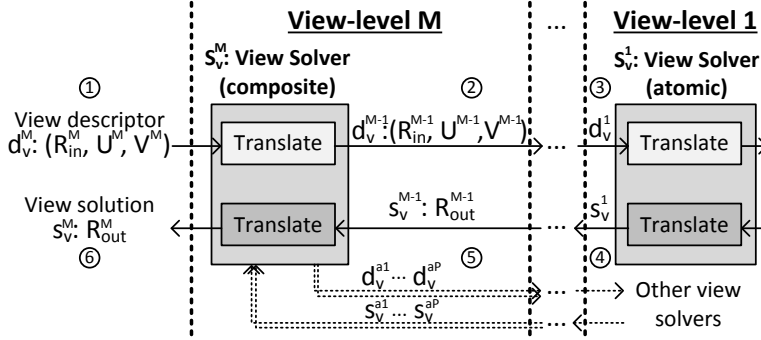


Figure 8.5: The multi-level abstraction workflow to abstract the details of the problem formulation (and solving)

WITH `sudoku_composite_solver()`

As seen in the query, the composite solver hides all the details of problem formulation, solving, and solution reverse transformation (see Section 8.1). Moreover, it allows intuitively describing an additional constraint requiring that the Sudoku matrix element (2,4) should not be equal to 4. As the matrix is in decimal (and not binary) format, this description is more convenient for the user. Internally, the Sudoku composite view solver involves the discussed view solver (*solverlp*), which we now term an *atomic* view solver as it performs no further transformations to the view-based format, but instead, performs direct translations to and from the relational format as explained in Section 8.3.1.

Composite view solvers used in SolveDB support optimization problems that can be mapped (or relaxed) to other problems supported by existing (composite or atomic) view solvers. Figure 8.5 shows the multi-level abstraction workflow encountered when a composite view solver S_v^M (e.g., *sudoku_composite_solver*) utilizes the solving capabilities of other existing view solvers (e.g., *solverlp*) from the levels 1 to $M-1$ to transform the descriptor d_v^M into a solution s_v^M . Here, S_v^M translates the descriptor d_v^M (compact) into the descriptor d_v^{M-1} (detailed). The processing of d_v^{M-1} with S_v^{M-1} might lead to intermediate view descriptors d_v^{M-2}, \dots, d_v^1 to be processed with the view solvers S_v^{M-2}, \dots, S_v^1 . Among these solvers, S_v^1 is an atomic view solver (such as *solverlp*). When S_v^1 finds the solution s_v^1 , it is propagated through the levels 1 to M by all respective solvers, resulting in S_v^M .

Alternatively, a composite solver (like FS in Section 8.8) might apply a heuristic, as part of which (traditional) optimization problems needs to be solved. Here, the capabilities of multiple view solvers are utilized through view descriptors $d_v^{a1}, \dots, d_v^{aP}$ leading to the intermediate solutions $s_v^{a1}, \dots, s_v^{aP}$.

To summarize, SolveDB supports *composite* view solvers letting users employ simpler constructs to formulate optimization problems or solutions in desirable formats (e.g., the decimal Sudoku output relation), employing the multi-level abstraction workflow is to produce an output relation.

8.3.3 Relational solve operator

The presented two part workflow involves a chain of solvers to process a particular view descriptor d_v . Often, several choices of (composite/atomic view, relational, and physical) solvers exists to establish such a chain. SolveDB aims to choose such solvers automatically, e.g., by matching the schema of R_{in} against those supported by the view solvers (see more in Section 8.6). When this is not possible, or to override the SolveDB choice, the user can explicitly specify all or some of the solvers to be used when processing d_v . To encompass the steps of solver selection and the resulting two-part multi-level workflow, we employ the *relational solve operator*, defined as follows:

Definition 8.2. Let \mathbb{S}_P be a so-called relational solve operator mapping the view-based descriptor $d_v (R_{in}, U, V)$ into the view-based solution $s_v (R_{out})$ such that $s_v = \mathbb{S}_P(d_v)$. Here, P is a set representing all or some of the solvers (composite/atomic view, relational, and physical solvers) to be used when processing d_v , along with configuration parameters for each solver. Every $p \in P$ is a pair $(name, p)$, where *name* indicates the solver name and *p* indicates additional parameter-value pairs to be passed to the solver to control the instance building and solving process (like in Definition 8.1).

8.4 Solve Query Formulation

We now show how Challenge 8.2 is addressed in SolveDB. First, we explain how SQL is extended to define a view descriptor and the predicate of the solve operator. Then, we show how user-defined views (V) from view descriptors can be formulated in a user-intuitive manner.

8.4.1 Extending SQL for optimization problems

As exemplified earlier, SolveDB extends SQL with a new SOLVESELECT clause with the following syntax:

```

SOLVESELECT col_name [, ...] IN ( select_stmt ) AS input_alias
[MAXIMIZE | MINIMIZE ( select_stmt ) ]
[SUBJECTTO ( (select_stmt) [, ...] ) ]
[WITH solver_name [. ...][ (param[:= expression] [, ...])]

```

The clause defines a view descriptor $d_v = (R_{in}, U, V)$ and the predicate P of the solve operator \mathbb{S}_P . Specifically, the keyword **SOLVESELECT** is succeeded by the list of attribute names representing unknown variables (U). The first select statement (*select_stmt* between **IN** and **AS**) defines the input relation (R_{in}) with the alias *input_alias* assigned. This alias is used to reference the input relation in all subsequent select statements in the **MAXIMIZE**/**MINIMIZE** clause and the **SUBJECTTO** block. This clause and the block are optional, and they serve as a “syntactic sugar” to define V_1, V_2, \dots, V_L in the sequence V . The mutually exclusive **MAXIMIZE** and **MINIMIZE** define V_1 or V_2 which specify one or more objective functions to be maximized or minimized, respectively. The **SUBJECTTO** block defines V_3, V_4, \dots, V_L and represents all remaining optimization model elements such as constraints. Depending on the solver, each $V_1, V_2, \dots, V_L \in V$ in **SUBJECTTO** is specified in either a solver- or a user-friendly way, which we elaborate in the next section. Finally, the **WITH** clause represents P and specifies names (*solver_name*) of all solvers to be involved when processing d_v . Solver names are given in the order such that names of *composite view solvers* are succeeded with the names of *atomic view solvers* and so on. The specified solvers can optionally be configured by providing the list of parameter-value pairs, where the value is either undefined or computed according to a user-specified SQL expression.

The **SOLVESELECT** clause has the semantics of the relational solve operator \mathbb{S}_P . As the clause results in a relation, it can be used as a sub-query in larger and more complex SQL queries, as shown in Section 8.1.

8.4.2 User-friendly view descriptors

As mentioned earlier, a view solver S_v poses requirements on how view definitions $V_1, V_2, \dots, V_L \in V$ from the view descriptor $d_v (R_{in}, U, V)$ should be specified and it is up to the solver developer to decide what V_1, V_2, \dots, V_L actually represents and how they are formulated in the **SOLVESELECT** clause. In one extreme, S_v can be a simple “wrapper” over a relational solver such that the view definitions V_1, V_2, \dots, V_L specify the relational solver input, i.e., $d_r = R_1, R_2, \dots, R_N$ where $N = L$. In the other extreme, S_v can be specialized for a very narrow class of problems and, by using the 3-tuple (R_{in}, U, \emptyset) , it can build a (relational or view) descriptor only from data in R_{in} , as in the example of *sudoku_composite_solver*. In the first case, the view solver allows utilizing the full solving capability of a relational (and physical) solver, but requires the detailed (and potentially complex) specification of the relational input d_r . In the second case, the solver uses a very simple input (R_{in} and U only), but is only targeted for a narrow class of problems to be instantiated from R_{in} . To address the limitation of these two

extremes and to establish a common user-friendly formulation of views in V , SolveDB uses an additional level of indirection over R_{in} .

In SolveDB, each V_i , $i = 1..N$, represents a view (query) over a dummy relation R_{in}^{alias} , where R_{in}^{alias} is an alias for R_{in} (*input_alias*). This separation of V_i and R_{in} is exploited in SolveDB to introduce an additional level of indirection, in which unknown variables in R_{in} are substituted with values of a user-defined data type that is specific to the solver. This data type offers a set of operators that are used to assign and constrain unknown variables. The set of supported operators depends on the capabilities of the solver, and, among others, may include both common comparison ($<$, $<=$, $=$, $>=$, $>$), equality ($=$), negation ($!=$), and similarity (\sim), and specialized (e.g., *is_instance*, *belongs*) operators.

Example 8.2 (Substitution of Sudoku relation attributes). *The attribute `val` in the query “`SELECT val!=4 FROM r WHERE row=2 AND col=4`” from Section 8.3.2 is used to reference unknown variables as if they were variables of the mathematical optimization problem. When processing such a query (as we explain later), SolveDB transparently substitutes the `val` attribute with the attribute of a solver-specific data type supporting the `!=` operator to bind unknown variables with concrete values from a database (or constants).*

We term this *substitution-based* formulation. It is intuitive, user-friendly, and yet powerful as it allows mixing unknown variables with known variables (from other attributes) when constraining and binding data to unknown variables. Depending on the needs, all or some of the capabilities of underlying (view or relational) solvers can be exposed to users through views (V) employing substitution-based formulation.

8.5 Solve Query Processing

We now show how SolveDB addresses Challenge 8.3. We first present the overall SolveDB solve query processing workflow, then focus on SolveDB-compliant view solvers offering *substitution-based* formulation, and present the common workflows of atomic and composite view solvers.

8.5.1 Overall solve query processing workflow

Like a standard DBMS, SolveDB prepares, optimizes, and executes a solve query as a single relational workflow involving the standard relational operators and the *relational solve operator*. To prepare (establish) a solve query, SolveDB performs the following actions for each SOLVESELECT clause:

1. Forms a view descriptor d_v and a predicate P (Section 8.4.1).

2. Establishes a valid chain of solvers (Section 8.3).
3. Instantiates and inlines the two-part multi-level translation workflow according to the solver implementation.

The formation of d_v and P is straightforward. To establish the chain of solvers, SolveDB uses P and d_v to look-up in a *solver catalogue* containing information about all registered (composite/atomic view, relational, and physical) solvers and then chooses (if possible) a subset of those to be used when processing d_v . For choosing, SolveDB relies on a *solver advisor* (explained in Section 8.6) or performs a simple matching of the names and data types of the unknown variable attributes and the operators in the views (V), against those supported by the solvers. When no or multiple solver choices are feasible, SolveDB reports an error and the user is required to explicitly specify (in the WITH clause) the solvers to be used. To instantiate the translation workflow, SolveDB consults the solver catalogue and then recursively unfolds the individual workflows of each (composite and atomic) view solver in the chain, resulting in a detailed relational workflow, which is then inlined (embedded) into the overall solve query workflow. This process is similar to *SQL function inlining*, reducing function call overhead and allowing the query optimizer to “see inside” the function. Once the complete solve query workflow is established, it is then optimized (as explained in Section 8.6) and executed.

Individual relational workflows of view solvers can be very different and complex depending on the type of an optimization problem and on if the solver is atomic or composite. As SolveDB encourages (but is not restricted to) solvers supporting *substitution-based* formulation (see Section 8.4.2), we now present the common workflows for the *substitution-based* atomic and composite view solvers used by SolveDB.

8.5.2 The workflow of an atomic view solver

As explained in Section 8.3.1, atomic view solvers expose the capabilities of relational solvers through view descriptors. We now present the common workflow of atomic view solvers supporting the substitution-based formulation. To exemplify relevant concepts, we explain how a general-purpose LP view solver can be designed to process the solve query of the binary Sudoku problem (see Section 8.1).

The workflow to transform $d_v (R_{in}, U, V)$ to $s_v (R_{out})$ is shown in Figure 8.6. For each view definition $V_i \in V$, the atomic solver S_v^a establishes the three-level nested views over R_{in} , namely R_{in}^o , R_{in}^s , and R_{in}^m to represent (build) the input of the relational solver S_r . Different view solvers might employ various alternatives of the views R_{in}^o , R_{in}^s , and R_{in}^m . Some of these alternatives are depicted in Figure 8.7.

The *order level* view R_{in}^o establishes the logical order of tuples in R_{in} so that unknown variables are referenced consistently when building the problem instance from R_{in} as well as when embedding the solution back to R_{in} . For simplicity of presentation, we introduce an explicit *order* attribute with unique integer values (see *order* in Table 8.3). To establish order in practice, any existing single or multi-attribute key would suffice.

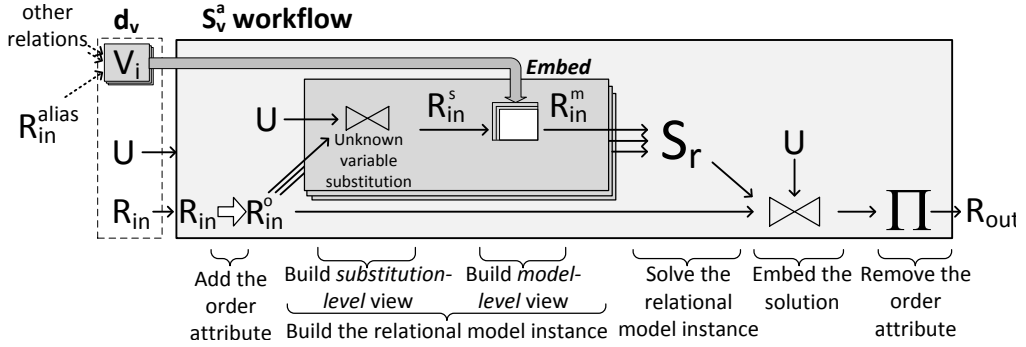


Figure 8.6: The workflow of an atomic view solver transforming a view descriptor into an output relation

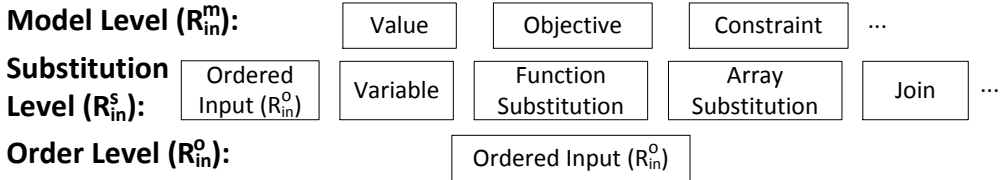


Figure 8.7: The hierarchy of views over R_{in}

To reference the variables, (logical) variables numbers are employed, and these are computed from the values of the order attribute for every $u \in U$.

The *substitution level view* R_{in}^s represents R_{in}^o , but with solver-specific values embedded at the unknown variable positions (at every $u \in U$).

Example 8.3 (Substitution level view of the LP solver). In the *Sudoku* case, the LP solver builds a view R_{in}^s as shown in Table 8.3 by applying a solver-specific function (UDF) $f_{LP} : \mathbb{Z} \rightarrow \text{LpExp}$ that maps unknown variable numbers to instances of the LP solver-specific data type *LpExp*. This data type represents the expression of unknown variables in the linear combination $a_1v_1 + a_2v_2 + \dots + a_tv_t$, where a_1, a_2, \dots, a_t are numerical constants and v_1, v_2, \dots, v_t are numbers of unknown variables. The *LpExp* type supports various linear operations such as expression addition, scalar multiplication, negation, etc., all together offering the rich formulation of expressions to be supported by the LP solver.

In the general case, R_{in}^s may have values that are (1) same as in R_{in}^o (Ordered Input) or are (2) the numbers of unknown variables (Variable), (3) values obtained from an array (Array) or another relation by joining a table on unknown variable numbers (Join),

Table 8.3: A substitution level view (R_{in}^s) over the binary Sudoku matrix

order :: integer	row	col	val	giv	sel :: LpExp
1	1	1	1	0	(1.[1])
2	1	1	2	1	(1.[2])
...
9	1	1	9	0	(1.[9])
...
729	9	9	9	1	(1.[729])

or (4) values returned from a function (UDF) applied on unknown variable numbers (as in the LP solver case) or a whole tuple in R_{in}^o (Function Substitution).

The *model level* view R_{in}^m represents a relation to be used as part of the relational solver input, d_r . The view R_{in}^m is established from V_i by substituting R_{in}^{alias} (the alias for R_{in}) in V_i with the substitution level view R_{in}^s . As a result, all operations with unknown-variables in V_i will be transparently applied on the instances of the solver-specific data type, e.g., *LpExp*. For example, suppose that *LpExp* supports an aggregation operator *sum* on *LpExp* instances as well as the equality operator $=$: $LpExp \times \mathbb{R} \rightarrow LpExp$ allowing to bind an instance of *LpExp* with a real number. Then, V_i from Section 8.1, formulated as “**SELECT sum(sel)=1 FROM in_b GROUP BY val, row**”, leads to R_{in}^m shown in Table 8.4. This relation represents the 81 (out of 243) binary Sudoku constraints (from Figure 8.2) ensuring that each Sudoku matrix column has no duplicated digits. In the general case, R_{in}^m characterises various optimization model components and may represent, for example, an objective function (Objective), constraints (Constraint), or pre-assigned values (those that are not NULL) of variables in R_{in} (Value).

In the workflow, all model level views, produced from $V_i \in V$, $i = 1..|V|$, represent the relational input (d_r) to be used by the relation solver S_r . The solution from S_r in the relational format (*var_nr*, *value*) is embedded back to R_{in}^o at unknown variable positions by building another substitution level view (of the *Join* type from Figure 8.7) to join two relations on variable numbers. Finally, the projection is applied to remove the order attribute (order) and to produce the output relation R_{out} in the schema of R_{in} .

Order, *substitution*, and *model* level views are standardized within SolveDB and used by different view solvers through an API to realise various translation workflows of substitution-based view descriptors. The LP solver is such a solver, producing relational problem descriptors as linear expressions according to user-defined (substitution-based) views.

Table 8.4: A model level view (R_{in}^m over R_{in}^s) representing 81 constraints for each Sudoku matrix column to have no duplicated digits

column1 :: LpEqu
((1·[1]+1·[10]+1·[19]+...+1·[82]), 1)
((1·[91]+1·[100]+1·[109]+...+1·[154]), 1)
((1·[163]+1·[172]+1·[181]+...+1·[235]), 1)
...
((1·[657]+1·[666]+1·[675]+...+1·[729]), 1)

8.5.3 The workflow of a composite view solver

As explained in Section 8.3.2, a composite view solver hides solver-specific details of problem solving by offering high-level user-intuitive constructs to (re-)define model elements such as constraints. We now show how such a composite view solver can be specified in a single solve query using the introduced *order*, *substitution*, and *model* level views. Then, we elaborate and generalize the resulting composite view solver workflow.

Consider the Sudoku solve query from Section 8.3.2 invoking the composite view solver *sudoku_composite_solver*. Suppose that the solver uses a descriptor d_e^c (R_{in}^c , U^c , V^c) as input, and that the constraint formulated as “**SELECT** val!=4 **FROM** r **WHERE** row=2 **AND** col=4” (included in V^c) leads to a model-level view m (val , op , row , col) with a single tuple (4, '!=', 2, 4). Then, *sudoku_composite_solver* can be specified as the following solve query relying on m and R_{in}^c (in_d):

```

1 SELECT out_b.row, out_b.col, out_b.val FROM (
2 SOLVESELECT sel IN
3   (SELECT col, row, v AS val, (val=v) as giv,
4     NULL::boolean AS sel
5     FROM in_d, generate_series(1,9) AS v) AS in_b
6 SUBJECTTO
7   -- All inherent (internal) Sudoku constraints
8   (SELECT sum(sel)=1 FROM in_b GROUP BY val, row),
9   (SELECT sum(sel)=1 FROM in_b GROUP BY val, col),
10  (SELECT sum(sel)=1 FROM in_b GROUP BY val,
11    (col-1) / 3, (row-1) / 3),
12  (SELECT sel = giv FROM in_b WHERE giv),
13  (SELECT sum(sel)=1 FROM in_b GROUP BY row, col)
14  -- All user-defined (external) constraints
15  (SELECT sel = 0 FROM in_b, m
16    WHERE m.col=in_b.col AND m.row=in_b.row AND
17    m.val=in_b.val AND m.op = '!=')
18 WITH solverlp.cplex()
19) AS out_b WHERE out_b.val

```

Like the solve query from Section 8.1, this query defines the transformations of the initial and final Sudoku matrices ($in_d \rightarrow in_b$ and $out_b \rightarrow out_d$). Additionally, this query incorporates (external) constraints (user-specified in V^c and represented in m)

output from the input relation based on a descriptor. The descriptor is either provided in the query or built along the solving chain. After the workflow is prepared, it is optimized and executed. Atomic and composite view solvers supporting *substitution-based* formulation are preferred, therefore, SolveDB offers an API for easily realising them. Such a realisation leads to a specific translation workflow, which was generalized and elaborated in this section. In the next section, we focus on techniques to optimize the overall solve query workflow.

8.6 Solve Query Optimization

Solve queries can often be very expensive to process as they rely on (physical) solvers tackling NP-complete problems with no efficient algorithms to find solutions. SolveDB is not optimizing these algorithms, but instead tunes (optimizes) problem descriptor and solution transformation workflows and chooses solver and solver parameters for a particular problem instance leading to the best performance or result quality. We now present query optimization techniques both for one-time and repeating solve queries, thus showing how Challenge 8.4 is addressed in SolveDB.

As presented earlier, solve queries might lead to complex relational workflows due to the applied chains of solvers for each SOLVESELECT. To reduce the processing time of these, SolveDB first uses standard query optimization techniques employed by existing DBMSs. Furthermore, in some cases, SolveDB is able to *completely avoid* invoking an (expensive) solver or *select* a solver and/or its parameters to achieve either the best performance or result quality.

Consider a projection on the output of the solve operator \mathbb{S} discarding the attributes with unknown variables. In this case, the input relation can be used in place of the output relation thus avoiding the burden of (view, relational, and physical) solvers, leading to the following equivalence rule:

$$\Pi_A(\mathbb{S}_P(R_{in}, U, V)) = \Pi_A(R_{in}), \text{ if } A \cap U = \emptyset \quad (8.1)$$

For repeating solve queries, SolveDB uses materialization [99] of the solver result (view-based and relational), allowing to look-up the solution in a materialized view using a descriptor instead of invoking a solver. For some problems, incremental optimization algorithms [100] are used to update the solution when the underlying data changes.

Finally, when solver invocation cannot be avoided, SolveDB builds a (valid) chain of solvers by automatically choosing solvers and solver parameters, in addition to those specified by the user (in P from the WITH clause), to achieve either the best performance or result quality. This case leads to the following equivalence rule:

$$\mathbb{S}_{P_1}(d_v) \sim \mathbb{S}_{P_2}(d_v), \text{ if } P_1 \subseteq P_2 \quad (8.2)$$

To choose valid solvers, SolveDB matches the descriptor against those supported by solvers (details in Section 8.5.1). When multiple solver choices exist, SolveDB utilizes

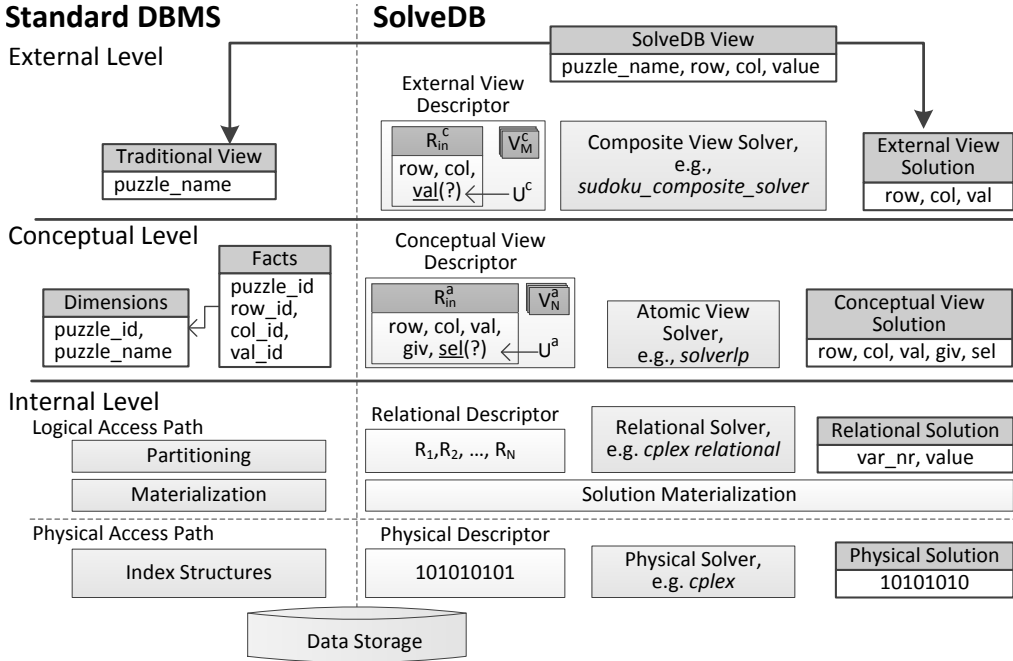


Figure 8.9: The architecture of SolveDB

a *solver advisor*, like DrAmpl [101], allowing to choose an appropriate solver for some types of problems. When multiple choices of solver parameter values exist, SolveDB uses default values or the values obtained by meta-optimization [102], which involves another solver to find the best parameter values for repeating solve queries.

8.7 Architecture of SolveDB

In this section, we address Challenge 8.5 and present the architecture of SolveDB supporting a variety of solvers.

SolveDB is based on the standard three-level ANSI/SPARC architecture (the left of Figure 8.9) with a number of additions on all three levels (the right of Figure 8.9). These additions, elaborated in Sections 8.3–8.6, allow separating the user’s view of the optimization problem from the way the problem is physically formulated and solved, like the user’s view of a database is separated from the way the database is physically represented by an ANSI/SPARC-based DBMS.

The additions at the internal schema level include a number of physical solvers

(e.g., *cplex* from Section 8.1) dealing with physical descriptors and solutions in internal (native) formats. Relational solvers from the same level hide the details of the internal representation and provide the relational representation of descriptors and solutions. Further on this level, the materialization of solve query results is used to provide quick answers for frequently repeating solve queries.

At the conceptual level, *atomic view solvers* (e.g., *solverlp* from Section 8.1) allow abstracting the details of the relational representation through high-level SQL-based constructs and an output relation. As these solvers use more abstract but powerful representations (schemas), the corresponding view descriptors and solutions are denoted as *conceptual*.

At the external level, *composite view solvers* (e.g., *sudoku_composite_solver* from Section 8.3.2) allow describing a problem and obtaining a solution in formats that are relevant to a particular user. Consequently, the corresponding view descriptors and solutions in user-specific formats are denoted as *external*. Finally, users define different customized (SolveDB) views over the underlying data and view-based solutions from the current level or the level below.

Similar to the *data independence* provided by standard DBMSs, SolveDB offers *solver independence* meaning that a user's external view descriptor can be processed by different (valid) combinations (chains) of *view*, *relational*, and *physical* solvers and a user does not necessarily need to know the combination that was actually used and how the problem was actually solved. A database administrator can change atomic view, relational, and physical solvers, while potentially affecting the query evaluation performance and the quality, but not the semantics of a user's problem solution.

8.8 Experimental Evaluation

We now demonstrate the versatility of SolveDB and experimentally evaluate its performance and usability/developer productivity. We show that integrating a solver into a DBMS allows significantly increasing the overall solving performance for I/O intensive problems. We also show that our proposed multi-solver approach and the substitution-based realization of a common language for queries and models are applicable for a wide range of optimization problems and offer significantly better tool usability/developer productivity for problems that are complex to specify. To show that SolveDB is *versatile*, we developed six proof-of-concept view solvers – two general-purpose atomic view solvers and four composite view solvers – and used them to solve eight optimization problems – four classical linear/mixed integer (LP/MIP) programming problems, three global optimization (GO) problems, and one domain-specific energy planning problem. To evaluate *performance*, we compared SolveDB against the existing tools by measuring (1) the solution quality (in terms of error or imbalance) and (2) the total time required to read initial data, solve the problem, and write the solution back to the database. Finally, to evaluate the *productivity/usability*, we measured the number of effective source

lines of code (eLOC) and compared SolveDB and existing tools in terms of the sizes of problem descriptors (incl. SQL-based data bindings) and solver implementations (where relevant). We now describe our PostgreSQL-based SolveDB implementation and present the settings and results of the experimental evaluation.

8.8.1 PostgreSQL-based implementation

We implemented SolveDB based on PostgreSQL 9.3.1. Our implementation employs a simplified solve query processing workflow where each SOLVESELECT clause is handled by a special user-defined function. The function establishes and invokes a solver chain in a single (execution) step, rather than in two (parsing and execution) steps, as explained in Section 8.5.1. Thus, the present implementation supports manual (but not yet automatic) *meta-optimization* and *solution materialization*.

To implement SolveDB, the PostgreSQL parser was extended to support the new SOLVESELECT syntax. Extensions (supported by PostgreSQL ≥ 9.1) were used to encapsulate the rest of the SolveDB functionality. The function handling each SOLVESELECT was included as a part of a so-called *SolveAPI* extension. The extension additionally defines the *solver catalogue* and a number of data types and functions allowing to develop atomic and composite view solvers as user-defined functions (in C/C++ or PL/pgSQL) in a common and simplified manner (see Sections 8.5.2–8.5.3). By utilizing *SolveAPI*, we implemented two atomic view solvers (abbr. as *LP* and *BB*) in C and packaged them as PostgreSQL extensions together with the underlying relational (logically integrated) and physical solvers.

The LP view solver delivers an exact solution after a finite sequence of operations by invoking a built-in physical solver from the GNU Linear Programming Kit (GLPK) for either LP or MIP problems. To process inputs and outputs of the physical solver, the LP view solver uses the technique described in Section 8.5.2. In contrast, the BB view solver, after a number of iterations, delivers an approximate solution for the class of black-box global optimization (GO) problems. To produce a solution, it uses 1 of 15 applicable physical solvers from the integrated SwarmOps optimization library [103], e.g., *Simulated Annealing* (SA) or *Particle Swarm Optimization* (PSO). The BB view solver prepares (the model view of) an objective function defined in the MAXIMIZE/MINIMIZE clause (see Section 8.4.1), and then a SwarmOps solver repeatedly re-evaluates the query to produce a new fitness value based on a candidate solution. The physical solvers from GLPK and SwarmOPS were patched to use PostgreSQL’s hierarchical memory management, and all the view and physical solvers run completely in the process (address) space of the DBMS.

SolveDB with all view solvers was compared to the existing tools by solving a number of optimization problems, covering all combinations of specification complexity and I/O intensity, as shown in Table 8.5.

Table 8.5: Optimization problems solved with SolveDB and existing tools

Nr.	Problem Name	$\hat{P}5, P5$ P1, P2	\uparrow Spec. complex $P5', P6$ \rightarrow Data intensive P3, P4	# of Vars
P1.1	Maximum independent set (619 edges)			50
P1.2	Maximum independent set (928 edges)			50
P1.3	Maximum independent set (1140 edges)			50
P2.1	Linear model fitting (10 regr. coef., 100 points)			111
P2.2	Linear model fitting (10 regr. coef., 1000 points)			1011
P2.3	Linear model fitting (10 regr. coef., 2000 points)			2011
P3.1	Sudoku 9x9 (23 initial values)			729
P3.2	Sudoku 16x16 (97 initial values)			4096
P3.3	Sudoku 36x36 (672 initial values)			46656
P4.1	Stigler diet (77 foods)			77
P4.2	Stigler diet (923 foods)			923
P4.3	Stigler diet (9923 foods)			9923
P5.1	Neural network black-box training (1x4x3x1 neurons, 10 points)			27
P5.1	Neural network training and output generation			27
P5.1	Meta-optimization of PSO solver parameters			31
P5.2	Neural network black-box training (1x4x3x1 neurons, 100 points)			27
P6	Energy planning – forecasting (3000 measurements) and scheduling (500 planning objects)			7983

8.8.2 Results of the experimental evaluation

Classical LP/MIP We used SolveDB with LP, GLPK, and *GNU R* to solve two LP problems, *Linear Model Fitting* (by least absolute deviations) and *Stigler Diet*, and two MIP problems, *Maximum Independent Set* (MIS) and *Sudoku*. These problems, denoted as P1, P2, P3, and P4 according to Table 8.5, were solved in three different sizes leading to 12 problem instances (P1.1, ..., P4.3). For each problem instance, we prepared a database with initial data and an empty table for the solution. Then, we described each problem, incl. SQL-based data bindings, as (1) a solve query for SolveDB, (2) a MathProg (mod) program for GLPK, and (3) an R program based on the *lpSolveAPI* library. The sizes of these descriptors (formulations) are given in Figure 8.10. Then, we solved each problem instance and measured *total time* and *solving time*, and calculated *I/O time* (*total time* - *solving time*). The results of the experiment are shown in Figure 8.11. This figure and Figure 8.10 show that SolveDB allows defining the problems much more compactly (approx. 1.5-3 times less code) and processing them much more efficiently (I/O time is 2-18 times smaller). As expected, the solving times (and total times for solving-intensive cases) of SolveDB and GLPK are similar as they share the same physical solvers. The R *lpSolveAPI* could not find solutions for the

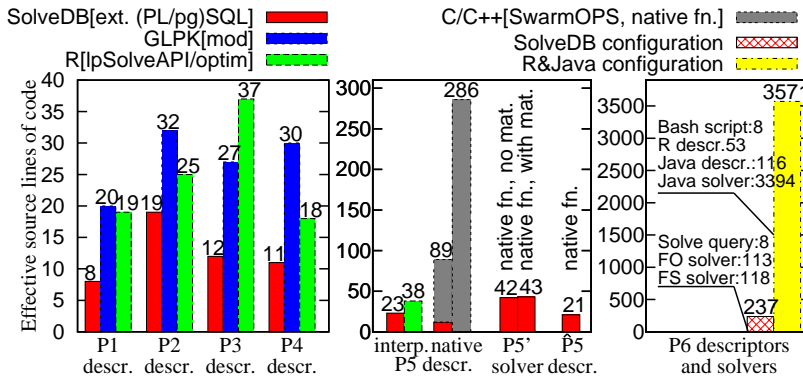


Figure 8.10: The sizes of implementations in effective source lines of code (eLOC)

Sudoku instances P3.1, P3.2, and P3.3 within 10 hours (increasing the number of given Sudoku digits from 23 to 62 in P1.1, yielded a solution after 11.6 sec, denoted by R*).

Black-box optimization We used SolveDB with BB (PSO), SwarmOPS (PSO), and R to train a neural network with a sigmoidal activation and four and three nodes in the two hidden layers, respectively. To train the network, we formulated an optimization problem (derivative-free), denoted as P5, minimizing the sum of squared error (SSE) of the training data. To solve the problem, we considered two datasets with 10 and 100 data points (P5.1 and P5.2 respectively) and two solving configurations relying on *interpreted* and *native machine* code, respectively. For the *interpreted* variant, we defined the problem as (1) a solve query using a recursive SELECT in the MINIMIZE clause and (2) an R program using the *optim* function for general-purpose optimization. For the *native* variant, we defined the problem as (1) a solve query specifying (in MINIMIZE) the use of a C function (from an external library) for computing network output based on provided network input and neuron weights, and (2) a C++ program relying on SwarmOPS and ODBC. Then, by using these two configurations and default solver parameters, we solved the two problem instances (P5.1 and P5.2) 6 times by varying the total number of solving iterations and measured the average resulting error (SSE) and total time. The results are shown in Figure 8.12(a). We see that the C++ implementation performs somewhat better (in terms of error) compared to SolveDB using the native function, but at a very substantial expense of usability/productivity (286 versus 89 lines). Compared to R (with *optim*), SolveDB (with BB) performs (approx. 30-100 times) better, considering that it repeatedly executes the query plan of a (prepared recursive) objective query, which can further be optimized (future work). In both configurations, SolveDB problem formulations have substantially less code (23 versus 38 and 59 versus 286 lines).

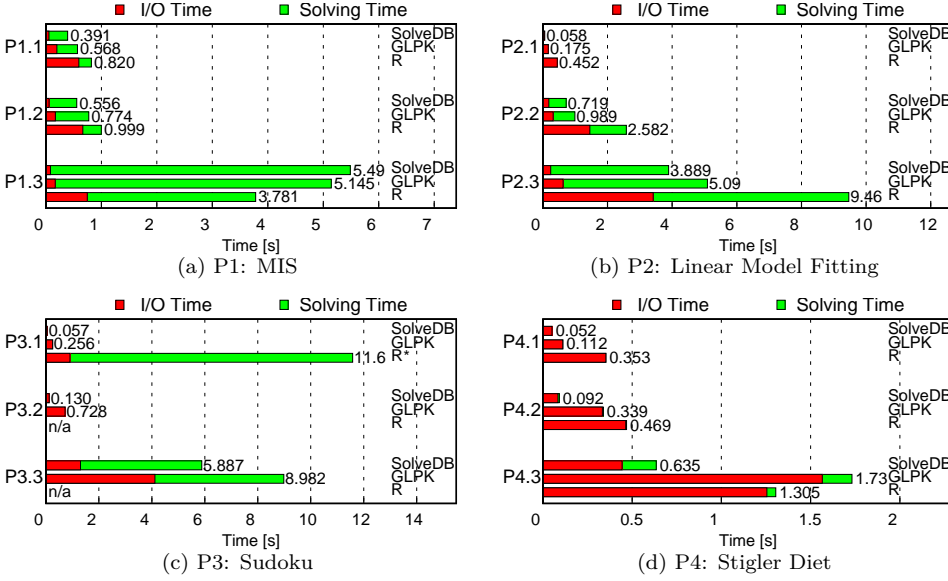
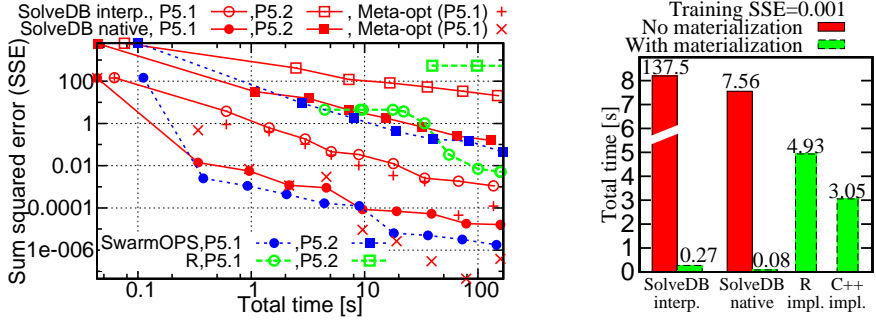


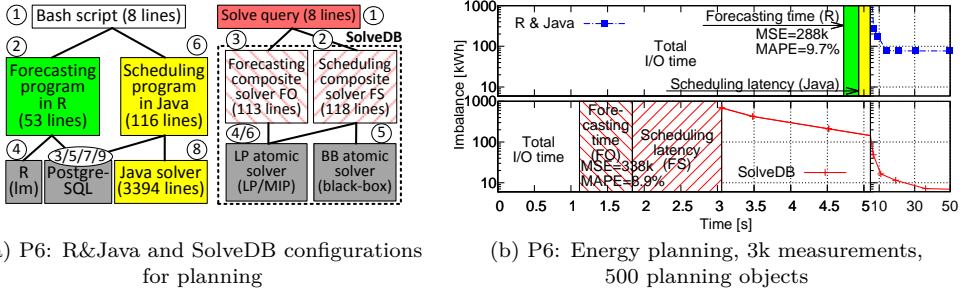
Figure 8.11: The performance of SolveDB, GLPK, and R when solving LP/MIP problems

Materialization/meta-optimization We used P5.1 as a basis for evaluating the *materialization* and *meta-optimization* query optimization techniques (see Section 8.6). We considered the joint problem of both (1) training a neural network (P5.1) and (2) computing network output values based on test inputs and optimized weights. To solve this joint problem, denoted as P5'.1, we built two composite view solvers both taking test inputs as the input relation and producing network outputs as the output relation. The first solver solves P5.1 each time it is invoked. In contrast, the second solver uses a *materialized view* over the corresponding SOLVESELECT for P5.1. As seen in Figure 8.12(b), when materialization is used, the total P5'.1 solving time is reduced by 2–3 orders of magnitude when training with $MSE = 0.001$ and using 1000 input (test) points, both for *interpreted* and *native* configurations. Compared with existing tools (the *R* and *C++ impl.* bars), computing network outputs using SolveDB takes approx. 11–62 times(!) less time compared to reading materialized weights and network inputs to the external tools and writing outputs back to the database. As a one-time investment which could pay-off after a number of view refreshes, we considered the problem of *meta-optimization*, denoted as $\hat{P}5.1$, for tuning P5.1 solver parameters for smaller error (and time for a desired error). We formulated $\hat{P}5.1$ as a nested solve query with the inner SOLVESELECT for P5.1 and the outer SOLVESELECT for PSO solver parameter $(S, \omega, \phi_p, \phi_g)$ tuning. We used SolveDB with BB and *Local Unimodal Sampling* [103]



(a) P5: Neural network black-box training, 10/100 points (b) P5.1: Computing 1k outputs

Figure 8.12: The performance of SolveDB, R, and C++ implementations when solving global optimization problems



(a) P6: R&Java and SolveDB configurations for planning

(b) P6: Energy planning, 3k measurements, 500 planning objects

Figure 8.13: Energy planning software configurations (a) and performance (b) in each configuration

with standard parameters as an overlying meta-solver, which was run 5 times for 100 iterations, solving P5.1 6 times in each iteration (3000 times in total). The meta-optimization results for each of the P5.1 SolveDB solving instances seen in Figure 8.12(a) show that meta-optimization is very effective (approx. 10-100 times smaller error) for time-consuming (above 10 seconds) cases with larger numbers of P5.1 solving iterations, while being very easy to specify in SolveDB.

Energy planning in MIRABEL Lastly, we used SolveDB to solve the BRP's energy planning problem, where the objective is to balance supply and demand from electricity producers and consumers according to hourly load forecasts for a 24 hour horizon. This problem inherently requires solving complex interlinked energy *forecasting* and *scheduling* sub-problems (see Section 3.6). We forecasted household consumption using a multi-equation EGRV model [76] which produces accurate hourly load forecasts based on historical power measurements and temperature forecasts used as external influences.

To balance the (inflexible) household consumption, we used joint loads from flexible consumers and producers, e.g., *electric vehicles* or *gas-turbines*. To represent loads of a single flexible consumer (and producer), we used flex-offers (aka. *unit flexibility models*). By choosing concrete amounts and a time shift, i.e., by scheduling, a flex-offer becomes a flex-offer assignment (aka. *unit prescription model*), defined as a time series (see Section 2.1). The objective is to schedule flex-offers so that all time series (incl., forecasted) add up to a (balanced) time series with (close to) zero values. To forecast loads and schedule flex-offers, we implemented two composite view solvers based on PL/pgSQL, *SolveAPI*, and SOLVESELECT: a *forecasting solver*, abbr. FO, and a *scheduling solver*, abbr. FS. The FO solver uses the LP solver to minimize least absolute deviations (like in P2) to estimate EGRV model parameters and then produces forecasts. The FS solver uses a heuristic technique, where the BB (with SA) and LP atomic view solvers find time shifts and the amounts for all planning objects, respectively. The solvers are interlinked using the following SolveDB query, previously denoted as *prescription query* (Section 4.3):

```
SOLVESELECT fa IN (SELECT fo::flexoffer, NULL::flexofferassignment AS fa
                     FROM flexoffers) AS t
SUBJECTTO (SELECT is_instanceof(fa, fo) FROM t),
           (SOLVESELECT load IN (SELECT time, load
                                   FROM hist_load) AS s
           SUBJECTTO (SELECT time, temp FROM temp_data)
           WITH solverFO)
WITH solverFS(rndseed:=12345, sn:=3176)
```

For solving the planning problem, we also considered the traditional configuration, abbr. R&Java. We implemented a Bash script, an R program (based on the model fitting function *lm*) to realize EGRV forecasting, and a Java program based on a third-party solver, previously used in the aggregation experiments (see Section 6.6.2). This solver realizes an evolutionary flex-offer scheduling technique [52] in Java. The R&Java and the SolveDB configurations together with component dependencies (lines), sizes, and invocation order (numbers in circles) are shown in Figure 8.13(a), respectively. We evaluated the performance of these configurations using 3000 (load/temperature) measurements and 500 planning objects representing supply and demand loads with complex flexibility patterns, and the results are shown in Figure 8.13(b). As seen in Figure 8.13(a-b), SolveDB required approx. 15 times (237 versus 3571 lines!) less total solver and descriptor code for solving the planning problem. It also allowed to obtain comparable forecasts ($10\% \pm 2$ of Mean Absolute Percentage Error) and a better schedule (plan) in less time, due to substantially reduced I/O time (>4 times!) and a higher-latency scheduling technique providing better immediate (first) solution due to exact solving with LP. It can be seen that forecasting-alone performs better in R (due to the efficient *lm*), but, due to *solver independence* (see Section 8.7), the FO solver could easily be improved by substituting the underlying (reused) LP solver with, e.g., the least-square solver used in R (*lm*).

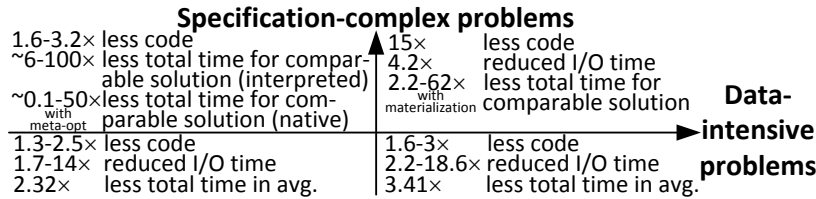


Figure 8.14: The summary of experiment results

The results are summarized in Figure 8.14 and they show that SolveDB is a competitive and versatile problem solving system offering a significantly increased (more than one order of magnitude less code) developer/user productivity, tool usability while at the same time providing comparable, or in many cases much increased, overall performance.

8.9 Related Work

The advantages of an integrated data management and problem solving system have already been recognised [104,105], but previous efforts were only at the conceptual level, and did not address the issues of designing, realising, and evaluating an efficient and easy-to-use integrated system suitable for different classes of optimization problems.

Most previous efforts of bringing problem solving capabilities into a DBMS focus on LP/MIP problems. The approaches supporting the block-schematic view [106] of a LP/MIP problem require users to specify either a stored procedure [107] or multiple SQL views [108] to define the “blocks” of a matrix, which is directly used by the simplex method [109] (one particular physical solver). This problem formulation (in the solver format) is very inconvenient for the user. In contrast, SolveDB (with LP) uses this representation only at the intermediate translation step when processing algebraic problem descriptors defined as solve queries and does so transparently to the user. SQLMP (SQL for Mathematical Programming) [110] offers an SQL-like syntax to define objectives and constraints and extract data from the database, similarly to a solve query. However, unlike the SOLVESELECT clause, SQLMP’s COMPUTE statement cannot be nested or used in a single query (e.g., Sudoku query in Section 8.1) to transform data and solution between user-convenient formats. A similar limitation is encountered in DGQL [111] (Decision Guidance Query Language), where each problem formulation leads into an excessive number of SQL views. These must be specifically annotated to express the optimization semantics to enable the translation, by a pre-defined reduction procedure, into the solver format. In comparison, SolveDB problem descriptors are processed using a DBMS query processor and the translation procedure can be extended and customized, depending on the problem type or the user’s perspective of it. Most importantly, SQLMP and DGQL only support LP/MIP problems, and they mainly

propose just the query languages, with little or no consideration of query processing, optimization, and DBMS integration, and no significant experimental evaluation, which are all considered by SolveDB.

Existing in-DBMS analytics tools and techniques, including *MADlib* [112], *Oracle Data Mining*, *Microsoft SQL Server Data Mining*, and *statistical* [113] and *forecasting* [94] methods, either implement or rely on specific (non-linear) problem solvers such as *simulated annealing* or *conjugate gradient*. Similarly to SolveDB, Bismarck [114] aims to unify the architecture of these in-DBMS tools and techniques. Bismarck does this by exploiting that a good deal of common data analytics tasks can be formulated and solved as convex optimization problems, and to this end proposes an efficient *incremental gradient descent* (IGD) solver based on user-defined aggregates. In comparison, SolveDB is much more general, as it supports all these types of physical solvers and exposes the capabilities (but not the complexities) of the various solvers to the query level so that database users and developers can deal with almost any kind of an optimization problem in a simplified, relational, and consistent way.

8.10 Summary and Discussion

In this chapter, we focus on the PRESCRIPTIVECPS agent in one of the decision making (*global prescription*, *aggregator*, or *disaggregator*) roles and, for the realization of their *solve*, *solve^R*, and *solve^M* operations (see Section 3.3), we encourage the use of SolveDB – the first DBMS integrating data management and problem solving capabilities. SolveDB allows users to easily define and solve optimization (decision) problems from many different problem classes (e.g., linear programming). Users can define, solve, and process the solution of a problem using a single *solve query* with an intuitive SQL-based syntax, similar to mathematical notation. Solve queries can be nested and mixed with traditional database queries, allowing to define complex *prescription* and *prescription adaptation* queries (Section 4.3). To process such *prescription* and *prescription adaptation* queries or any other *solve query*, SolveDB uses one or more (of the many supported) SolveDB-compliant solvers (automatically or manually selected). Each solver supports a multi-level translation workflow that provides the problem formulation and solution in formats that are convenient for the user rather than for the underlying solving technique. We presented the overall solve query processing workflow and elaborated on how SolveDB-compliant solvers – the general-purpose (LP/MIP) and the problem-specific (Sudoku) solvers – solve optimization problems. Finally, we discussed solve query optimization techniques and presented the ANSI/SPARC-based SolveDB architecture and our PostgreSQL 9.3.1 based SolveDB implementation. The experimental evaluation, considering eight database-based optimization problems – four LP/MIP, three black-box global optimization, and one domain-specific problem – showed that SolveDB significantly reduces the number of lines of code (by 1.3-15 times), I/O time (by 1.7-18.6 times), and total time (up to 62 times with optimizations applied) compared

with existing optimization software.

Future work on in-DBMS problem solving will investigate further solve query optimizations, including new types of solvers that incrementally update solutions, are massively parallel, support automatic differentiation, or compile solve query problem formulations into efficient code (e.g., JIT-compiled Java), offering much better performance for repeated queries.

Chapter 9

Flexibility and Prescription Model What-If Analysis in a DBMS

In this chapter, we consider a PRESCRIPTIVECPS agent in one of the decision making (*global prescription*, *aggregator*, or *disaggregator*) roles and focus on the “what-if” capability of the DBMS component used as part of the agent’s software system (see Chapter 4). Unlike in Chapters 7–8) where we introduce new DBMS technology, here we primarily use examples to demonstrate simplified powerful ways to analyse past, current, and future data based on flexibility and prescription models and so-called hypothetical *scenarios*, which, as we elaborate later, can be supported by the agent’s analytical DBMS. We start by introducing the concept of *what-if scenario* and reviewing related work in Section 9.1. Then, in Section 9.2, we propose a simple and intuitive SQL extension for being able to specify scenarios utilising SQL constructs. Then, in Section 9.3, we use the proposed extended SQL syntax to demonstrate advantages such scenarios bring when analysing measured/forecasted data, planning energy, and studying previous versions of data in the MIRABEL use-case. As we conclude in Section 9.4, hypothetical what-if scenarios bring simplicity to complex analytical activities.

9.1 What-If Analysis and Related Work

In a human-based (manual) decision-making or decision model adaptation (Requirement 9), users often want to understand implications of various human- or machine-made decisions, expected or hypothetical conditions, and realistic or artificial circumstances. To accomplish this, *what-if* analysis can be utilized.

Traditionally, the *what-if analysis* is used to simulate and inspect the behaviour of a complex system under some given hypotheses, called a *scenario*. In the what-if analysis, an analyst first formulates a scenario (e.g., as hypothetical data inserts, modifications, and deletes), then uses the scenario to derive a hypothetical “world”, and then explores the world by querying and navigation. There are few commercial tools capable of supporting what-if analysis (incl., *Excel spreadsheet*) and some papers [115, 116] describing relevant applications in different fields. The capabilities of DBMSs can also be utilized to perform what-if analysis in cases when access to a data source is either *unrestricted* or *restricted*.

In the **unrestricted case**, the what-if analysis can be performed using conventional means, e.g., by building a replica of a database and/or running multiple variants of a query and performing transactional updates (and then rollbacks) if needed. However, building replicas or performing updates/rollbacks are expensive and cumbersome. To avoid transactional updates, Heraclitus/HQL [117, 118] and SESAME [37] (1) rely on query rewriting techniques that rewrite (hypothetical) queries according to hypothetical update scenarios, and (2) investigate hypothetical query languages that combine querying with hypothetical updates, and their implementation over data warehouses.

In the **restricted case**, Caravan [119] considers a client-server approach, where the what-if analysis is performed using a resource-limited device (e.g., tablet) on the client-side, supported by a relational DBMS and a specialized query rewriting system on the server-side. When a user activates one of more of pre-defined scenarios on the resource-limited device, results are efficiently re-computed on the client-side using a parametrised data instance, termed a *provisioned autonomous representation* (PAR). Such data instances (PARs) are compiled on the server-side out of the data, initial analysis queries, and what-if scenarios.

In the context of PRESCRIPTIVECPS agent, a user (owner) of an agent has an unrestricted access to a data managed by the underlying analytical DBMS. Therefore, for a what-if analysis in PRESCRIPTIVECPS context, the query rewriting approach, pursued by Heraclitus/HQL and SESAME, is the most suitable and therefore will be further extended (with SQL notations) and used to demonstrate potential applications and the power of what-if analysis in the PRESCRIPTIVECPS context.

9.2 SQL Extension for What-If Scenarios

For our demonstrations, we now propose a simple and intuitive SQL extension over the existing Hypothetical Query Language [118] (HQL) for being able to specify what-if scenarios using the SQL constructs. Our proposed SQL extension comprises of the following commands/clauses:

- **CREATE SCENARIO** *name* **AS** (*cmd_1*; *cmd_2*; ...; *cmd_k*;) - the command creates a scenario termed *name* in a database. The scenario is specified using a sequence

of data manipulation language (DML) commands $cmd_1, cmd_2, \dots, cmd_k$, each of which is either of INSERT, DELETE, or UPDATE type. These commands describe how the state of the database is (hypothetically) modified to produce a new hypothetical database state (world).

- **CREATE SCENARIO** *name* **AS OF** sn_1, sn_2, \dots, sn_k - the command creates a composite scenario termed *name* in a database. It allows a simplified way of creating a scenario by sequentially composing individual DML commands of scenarios sn_1, sn_2, \dots, sn_k .
- **DROP SCENARIO** *name* - the command removes a scenario termed *name* from a database (if scenario exists).
- **SELECT ... FOR** sn_1, sn_2, \dots, sn_m - the clause specifies a selection (SELECT) query that is to be evaluated on a hypothetical database state obtained by sequentially applying (executing) commands of scenarios sn_1, sn_2, \dots, sn_k on the current database state (real or hypothetical). This statement has the semantics of the following hypothetical HQL query $(\dots((Q \text{ when } U_1) \text{ when } U_2)\dots) \text{ when } U_m$, where Q is a relational algebra query represented by the **SELECT** clause and U_1, U_2, \dots, U_m are update expressions (*ins, del*) represented by the scenarios sn_1, \dots, sn_k .

Here, the **CREATE SCENARIO** commands allows defining (atomic or composite) scenarios as a sequence of INSERT, DELETE, and UPDATE commands, similarly to how user-defined functions are defined. The **SELECT ... FOR** is similar to the traditional SQL SELECT statement. Unlike the latter, it allows querying data under the presence of hypothetical updates specified by scenarios. Like hypothetical HQL queries, it can be evaluated using *eager, lazy, or hybrid* approaches [118], requiring the full, no, or partial materialization of hypothetical physical state (or the corresponding delta), respectively. A DBMS evaluating such query ensures that a base data in a database is not affected by hypothetical updates.

Note, the **SELECT ... FOR** statement is similar (and complementary) to that of *time travel* queries. Time travel queries [38], first pioneered by Oracle with Flashback [120] and later standardized in SQL 2011 [121], allow navigating old versions of data as if an application (business) or a system time is (hypothetically) set to that specified in a query, e.g., as in **SELECT ... FOR SYSTEM_TIME AS OF '2014-01-01'**. Consequently, the time-travel functionality of existing bi-temporal databases can be seen as the special case of scenario-driven querying where application or system time is (hypothetically) updated (e.g., by pruning irrelevant tuple versions). Therefore, as shown later, we allow mixing hypothetical scenarios and time travel predicates (*BUSINESS_TIME* and *SYSTEM_TIME*) in a single **SELECT ... FOR** query.

The presented SQL extension as well as the discussed what-if and the time travelling functionality can potentially be supported by the agent's DBMS. In the next section, we use the proposed SQL extension to demonstrate what-if queries in MIRABEL.

Table 9.1: Tables for energy consumption measurements and flex-offers/flex-offer assignments

(a) hist_load table

time	load
...	...
12:00	10.9
12:15	11.3
12:30	12.4
12:45	13.3
13:00	12.1
13:15	11.6

(b) flexoffers table

id	fo	fa	sids
1	(startTime:21:00-07:00, values: [2-3, 3-4, 5-6, 1-3])	NULL	{}
2	(startTime:20:30-06:15 values: [3-4, 5-5.5])	NULL	{}
3	(startTime:21:00-04:30 values: [3-4, 5-6, 3-4])	(startTime:22:00 values: [3.5, 6, 3.4])	{1, 2}
...

9.3 What-If Queries in the MIRABEL Use-Case

An agent's DBMS with the proposed SQL extension for what-if scenarios can be effectively used to study implications of various decisions/conditions/circumstances based on flexibility and prescription models. We demonstrate this using the MIRABEL use-case.

9.3.1 Hypothetical scenarios in measurement analysis

Suppose that historical consumption load measurements are stored in a database table, shown in Table 9.1(a), and an analyst (BRP) wishes to study future measurements as if they (1) followed forecasts exactly or (2) were 10% higher than forecasts produced from 5% lower measurement values. The analyst would then create the following two scenarios, representing the two cases respectively:

```

CREATE SCENARIO con_follows_forecasts AS (
  INSERT INTO hist_load(time, load)
  SELECT time, load FROM (
    SOLVESELECT load IN (SELECT time, load FROM hist_load) AS s
    WITH solverFO ) AS measurements_with_forecasts
  WHERE time BETWEEN now() AND now() + INTERVAL '24_hours'
);

CREATE SCENARIO con_off_forecasts AS (
  INSERT INTO hist_load(time, load)
  SELECT time, 1.1 * load FROM (
    SOLVESELECT load IN (SELECT time, 0.95 * load FROM hist_load) AS s
    WITH solverFO ) AS measurements_with_off_forecasts
  WHERE time BETWEEN now() AND now() + INTERVAL '24_hours'
);

```

Here, the scenarios `con_follows_forecasts` and `con_off_forecasts` are based on *solve queries* relying on the *forecasting solver* (FO) introduced in Section 8.8. Note, to define

these scenarios, alternatively, *forecasting queries* over a pre-defined *time series view* for measurements (`hist_load`) can be employed (see Section 7.1.3). The presented two scenarios can be used, for example, to contrast day-ahead consumption measurements in those two hypothetical cases, using the following declarative *what-if query*:

```
SELECT f1.time, f1.load - f2.load
FROM (SELECT time, load FROM hist_load FOR con_follows_forecasts) AS f1,
     (SELECT time, load FROM hist_load FOR con_off_forecasts   ) AS f2
WHERE (f1.time = f2.time)      AND
      (f1.time BETWEEN now() AND now() + INTERVAL '24_hours')
```

The query results into a relation with differences in consumption loads in those two hypothetical cases. It is more compact and intuitive compared to the following equivalent non-hypothetical query relying on the *common table expression* (CTE) and the *forecasting solver* (FO):

```
WITH
  con_follows_forecasts AS (
    SELECT time, load FROM (
      SOLVESELECT load IN (SELECT time, load FROM hist_load) AS s
      WITH solverFO      ) AS measurements_with_forecasts
    WHERE time BETWEEN now() AND now() + INTERVAL '24_hours'
  ),
  con_off_forecasts AS (
    SELECT time, 1.1 * load FROM (
      SOLVESELECT load IN (SELECT time, 0.95 * load FROM hist_load) AS s
      WITH solverFO      ) AS measurements_with_off_forecasts
    WHERE time BETWEEN now() AND now() + INTERVAL '24_hours'
  )
SELECT f1.time, f1.load - f2.load
FROM (SELECT time, load FROM con_follows_forecasts) AS f1,
     (SELECT time, load FROM con_off_forecasts   ) AS f2
WHERE (f1.time = f2.time)
```

9.3.2 Hypothetical scenarios in energy planning

Hypothetical scenarios enable easy-to-use *prescription queries* (and *prescription adaptation queries*) allowing to study (potential) prescriptions (or their feasible changes) derived “on-the-fly” during the query evaluation (see Section 4.3). We now demonstrate *prescription queries* that are based on what-if scenarios.

Suppose that an analyst wishes to study potential flex-offer assignments resulting after the complex workflow that involves flex-offer aggregation, flex-offer scheduling, and flex-offer assignment disaggregation, performed by aggregator and BRP agents as presented in Section 2.2 and Section 3.6. Suppose flex-offer assignments and input flex-offers (used to generate these flex-offer assignments) are represented as composite (PostgreSQL) data types and stored as pairs in the database table `flexoffers`, shown in Table 9.1(b). Initially, flex-offer assignments are not available and, thus, values of the

`fa` attribute are set to `NULL`, as shown in the $id = 1$ and $id = 2$ cases in Table 9.1(b). When a flex-offer is aggregated, the value of the `sids` attribute is not `NULL` and it specifies an array with ids of flex-offers that were aggregated to produce the current flex-offer, as shown in the $id = 3$ case. The objective is to generate flex-offer assignments (values of the `fa` attribute) for all non-aggregated flex-offers (those with `sids IS NULL`). This can be encapsulated in the following composite scenario:

```
CREATE SCENARIO assignments_generated AS OF
    flexoffers_aggregated,
    flexoffers_scheduled,
    assignments_disaggregated;
```

Here, `flexoffers_aggregated`, `flexoffers_scheduled`, and `assignments_disaggregated` are individual (non-composite) scenarios specified below.

```
CREATE SCENARIO flexoffers_aggregated AS (
    INSERT INTO flexoffers (fo, sids)
    SELECT agg.fo, (agg.fo).sids
    FROM (SELECT reduceA(fo) AS fo
        FROM flexoffers
        WHERE sid IS NULL) AS agg;
);
```

The scenario `flexoffers_aggregated` specifies (hypothetical) insertions of aggregated flex-offers, which are produced from non-aggregated flex-offers (inner `SELECT`) using a user-defined aggregation function `reduceA` that realizes the lossy flex-offer aggregation ($reduce^A$), presented in Chapter 6.

```
CREATE SCENARIO flexoffers_scheduled AS (
    UPDATE flexoffers
    SET fa = s.fa
    FROM (SOLVESELECT fa IN (SELECT fo, NULL::flexofferassignment AS fa
        FROM flexoffers
        WHERE sid IS NOT NULL) AS t
        SUBJECTTO (SELECT is_instanceof(fa, fo) FROM t),
        (SOLVESELECT load IN (SELECT time, load FROM hist_load) AS s
        SUBJECTTO (SELECT time, temp FROM temp_data)
        WITH solverFO)
        WITH solverFS(rndseed:=12345, sn:=3176) ) AS s
    WHERE id = s.id;
);
```

The scenario `flexoffers_scheduled` specifies (hypothetical) updates of aggregated flex-offer assignments. To generate assignments, the *scheduling solver* (FS) and the solve query from Section 8.8.2 are used.

```
CREATE SCENARIO assignments_disaggregated AS (
    UPDATE flexoffers
    SET fa = (SELECT mapA(a.fa, a.fo, i.fo)
        FROM flexoffers AS i,
        flexoffers AS a
```

```

        WHERE (i.id = ANY (a.sids)) AND (i.id = u.id)
    FROM flexoffers u;
-- Assignments updated, now delete aggregated flex-offers
    DELETE FROM flexoffers
    WHERE sids IS NOT NULL;
);

```

Finally, the scenario `assignments_disaggregated` specifies (hypothetical) updates of non-aggregated flex-offer assignments. To generate assignments, a user-defined disaggregation function `mapA` is used. It deterministically converts aggregated flex-offer assignment into non-aggregated flex-offer assignments using the approach discussed in Section 6.3. Additionally, the pairs of aggregated flex-offers and their assignments are (hypothetically) deleted after flex-offer assignments are generated, as they were used just as an intermediate result.

The composite scenario `assignments_generated`, encompassing all the presented individual scenarios, can be used in various queries involving flex-offer assignments (prescriptions). For example, the following simple query computes the total expected balance between consumption and production resulting after the scheduling:

```

SELECT SUM(unnest(fa.values))
FROM flexoffers
FOR assignments_generated;

```

Here, the (PostgreSQL) library function `unnest` expands an array with energy values to a set of rows. Furthermore, the generated flex-offer assignments can be physically stored in a database (materialized) using the following query:

```

UPDATE flexoffers
SET fa = (SELECT fa
          FROM flexoffers as h
          WHERE h.fid = f.fid
          FOR assignments_generated)
FROM flexoffers f

```

As seen in the examples above, scenarios allow hiding the (complex) details of flexibility and prescription model processing, and offer simplified ways to ask what-if questions about, for example, (invariant) flexibility models and prescriptions that are forecasted or computed by optimization on-the-fly.

9.3.3 Time travelling in energy planning

The time travel feature [38] of temporal databases may further simplify the analysis of historical flexibility and prescription model instances. Suppose an analyst wants to compute the difference between expected and actually metered inflexible consumption for the last 24 hours. By utilizing both the time travel predicate (see Section 9.2) and the previously introduced `con_follow_forecasts` scenario, the corresponding query can be formulated as follows:

```

WITH forecasts_yesterday AS (
    SELECT time, load
    FROM hist_load
    FOR SYSTEM_TIME AS OF (now() - INTERVAL '24h_hours'), con_follows_forecasts)
SELECT time, (t.load - y.load) AS 'difference'
FROM hist_load AS t INNER JOIN forecasts_yesterday AS y ON t.time=y.time
WHERE time BETWEEN (now()-INTERVAL '24_hours') AND now()

```

As seen in the examples above, hypothetical *what-if scenarios*, when combined with *forecasting*, *optimization*, and *time travelling*, allow for simplified ways to ask questions about past, presence, and future.

9.4 Summary and Discussion

In this chapter, we demonstrated how a scenario-based what-if analysis involving flexibility and prescription models can be used to study implications of various human- or machine-made decisions, expected or hypothetical conditions, and realistic or artificial circumstances. For the demonstration, we used the MIRABEL use-case and, for being able to use SQL constructs, proposed a simple and intuitive SQL extension for in-DBMS what-if queries based on the existing *hypothetical query language*. The extended SQL together with the presented what-if and the time travelling functionality can, potentially, be integrated into the DBMS of an agent in one of the decision making roles (*global prescription*, *aggregator*, or *disaggregator*). The presented use-cases of scenario-based measurement analysis, energy planning, and time travelling witness the increased simplicity of the complex analytical activities involving past, current, and future data.

This chapter covered what-if query specification aspects only. Potential future work would investigate different approaches for materialising hypothetical database states (or state deltas) incurred by various scenarios, particularly those involving *forecast* or/and *solve* queries. Another future work would investigate *sessions*, where a particular what-if scenario is activated at the beginning of the session and then it persists until changed (with another scenario) or the session ends. During the session, standard SQL queries are to be used for querying a hypothetical database state.

Chapter 10

Flexibility, Decision, and Prescription Model Processing in Real-Time

In this chapter, we focus on the planning components of a PRESCRIPTIVECPS agent software system (Section 4.1) and show how it can be designed to offer the desired near real-time agent performance (Requirement 1). Our proposed real-time (RT) architecture of the planning component supports both *analytical* and *operational* workflows involving measurements and flexibility/prescription model instances and, for the real-time setting, enables many *storage* and *query processing* optimizations and offers useful features such as *model-specific*, *approximate*, and *subscription-based* queries. Results of initial experiments witness significant performance improvements with regards to (historical) measurement storage, flexibility model aggregation, and forecast model maintenance. The chapter is structured as follows. Section 10.1 discusses the limitations of the discussed DBMS with built-in analytics when processing *analytical* and *operational* workloads with (soft or hard) constraints over a processing time. Then, Section 10.2 proposes the architecture of the planning component and Sections 10.3–10.5 discuss optimizations enabled by this architecture. Finally, Section 10.6 presents experimental results and Section 10.7 contrasts the RT-ready agent software system to existing work and concludes the chapter. The content of this chapter is based on Publication [8].

10.1 Analytical and Operational Agent Workflows

As presented previously, a PRESCRIPTIVECPS agent software system processes two kinds of workloads: *operational* and *analytical*. Here, operational workloads transform

instances of a flexibility model into instances of a prescription model (Section 3.4). Analytical workloads, on the other hand, are submitted by (human) users to analyse and tweak the processes of planning and plan realization (Section 4.2). Operational workloads typically have (loose or strict) constraints on the processing time and often require low-latency and high-throughput data processing. In contrast, analytical workloads typically have less strict (or no) constraints on the processing time and often tolerate high-latency and low-throughput data processing.

Example 10.1 (Two types of workloads in the MIRABEL use-case). *In the MIRABEL use-case, a BRP agent conducts various data processing tasks with or without constraints on the processing time. Real-time grid balancing, for example, requires fast flex-offer processing to be able to meet user-specified deadlines (e.g., latest start time) as well as other deadlines such as electricity market closing times. Risk analysis, long term planning, and invoicing, in contrast, have loose (or no) constraints on the processing time.*

The presented DBMS with built-in (forecasting, optimization, what-if) analytics (Section 4.2) can be used for both analytical and operational workloads, but only those that have no constraints on the processing time. When time-bounded and/or low-latency and high-throughput workloads are involved, the presented DBMS is insufficient, and a new real-time engine for flexibility and prescription model processing is required. To address this, we now show how the planning-component of the agent software system (Section 4.1) can be transformed (extended) into such real-time processing engine allowing many optimizations. The extended planning component is inter-operable with the other agent components (incl., DBMS, control, GUI) and can provide new types of *model-specific*, *approximate*, and *subscription-based* queries suitable for real-time (RT) model processing and beyond this. We now present the design of such RT-ready planning component.

10.2 Planning Component Architecture for Real-time Processing

We design the software architecture of the planning component for (near) real-time processing of flexibility and prescription models (Requirement 1). The architecture is shown in Figure 10.1, additionally denoting the main (sub-)components involved when processing *operational* and *analytical* workloads, as well as the types of information exchanged with the control/GUI and the DBMS components (see Section 4.1). We now elaborate each (sub-)component individually and then discuss optimizations this architecture allows for real-time processing.

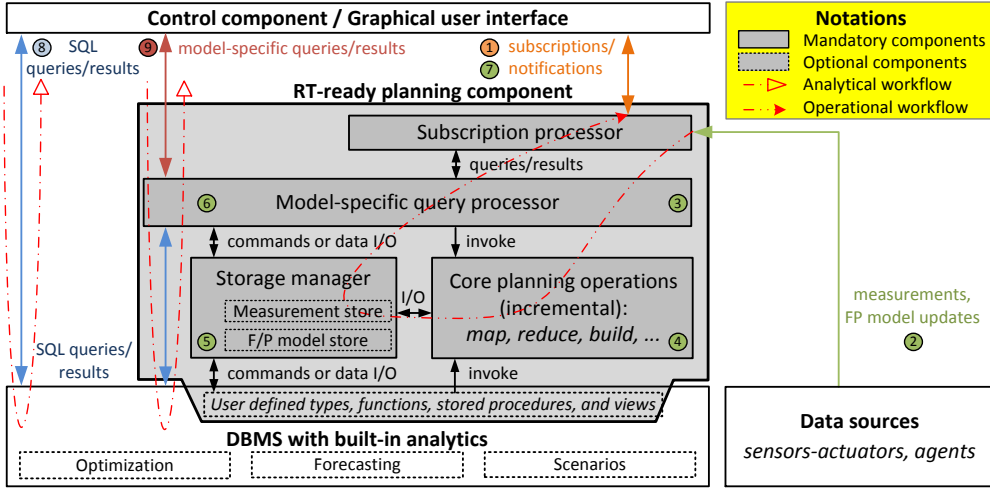


Figure 10.1: The planning component architecture for real-time (RT) processing

Core planning operations component includes agent-specific (or role-specific) realisations of required set of core planning operations such as *build*, *reduce*, and *solve* (see Section 3.3 for more). For real-time processing, incremental variants of these operations are adopted, e.g., like the incremental *reduce*^A presented in Section 6.5.

Storage Manager component provides a dedicated store for caching and/or short-term storage of latest (operational) measurements (see *measurement store*) and flexibility and prescription models – those that are not yet prescribed and expired (see *F/P model store*). The component offers direct data input and output (I/O) and may process basic I/O commands (e.g., *put/get*). It is realised as a main-memory heap, an in-memory DBMS, or a disk-based DBMS with a dedicated storage.

Model-specific query processor component offers the object-relational mapping (ORM) abstraction over the content of the storage manager (with a dedicated store) and the underlying analytical DBMS (elaborated in Chapters 7–9). The component processes so called *model-specific* queries which offer imperative and object-oriented constructs for managing instances of flexibility and prescription models. Due to object-oriented syntax, *model-specific* queries are often more convenient than SQL as they hide model representation and storage details, offer additional model-specific optimizations, and provide increased productivity of (human) operators monitoring, analysing, and managing the state of an agent in real-time.

Example 10.2 (A model-specific query for flex-offer retrieval). Suppose that the flex-offer Nr. 12345 (the instance of a unit flexibility model) needs to be retrieved from the underlying DBMS (see Chapter 5). To accomplish this, the following three

queries (at least three) have to be issued for retrieving flex-offer header, energy profile, and other related data scattered across multiple tables:

```
SELECT * FROM F_flexOffer WHERE flexOfferId=12345;
SELECT * FROM F_enProfileInterval WHERE flexOfferId=12345;
SELECT * FROM F_aggregationMeta WHERE flexOfferId=12345;
```

Instead, the following model-specific ORM-based query in the syntax of R can be used to retrieve the flex-offer:

```
unitFM <- getFlexOffer(12345);
```

The retrieved flex-offer (referenced by unitFM) can then be analysed like a traditional class object, e.g., by invoking methods as in `unitFM.getEarliestStartTime()`.

Subscription processor component allows (un-)subscribing (from) to notifications that are generated every time the result of a so-called *subscription-based* continuous query changes by more than a user-specified threshold. To produce such notifications the component processes measurements and (flexibility and prescription) model updates received from sensors-actuators, successor/predecessor agents, or external data sources.

Example 10.3 (A subscription-based flex-offer retrieval query). *The following continuous subscription-based query allows requesting flex-offers from the dedicated store and then be notified every time flex-offers in the dedicated store changes by more than 10kWh (from the previously reported flex-offers):*

```
NOTIFY ON energy_delta > 10000
IN getFlexOffersInDedicatedStore()
```

Notifications are caused by flex-offer inserts and deletes in the dedicated store (the storage manager), resulting after they are received from other agents or expire due to their inherent deadlines (e.g., latest assignment time or latest start time).

Furthermore, the core planning operations as well as the content of the dedicated store might be exposed to the underlying analytical DBMS, e.g., using DBMS extensibility features such as *user-defined types/functions/aggregates*, *stored procedures*, *views*, *view solvers* (see Chapter 8), or *scenarios* (see Chapter 9). This allows accessing latest (operational) measurements and model instances in various analytical workflows relying on traditional SQL queries, *forecasting queries* (see Section 7.1), *solve queries* (see Chapter 8), and/or *what-if queries* (see Chapter 9), and not only using *model-specific* queries.

We now discuss potential optimizations this architecture allows for real-time processing.

10.3 Data Storage Optimizations

The proposed architecture allows the following (traditional and novel) optimization techniques related to the incoming data storage: *caching*, *data locality*, *bulk-loading*, *measurement-model substitution*, and *model-compression*. We now discuss these optimization techniques.

Caching (traditional) allows making measurements and instances of flexibility and prescription models available for querying as soon as they are received from other (successor, predecessor, external) agents. Arriving data is first recorded in an intermediate dedicated store (see *storage manager* in Figure 10.1), while making data available for querying like in RiTE [122], before it gets materialized in a database.

Data locality (traditional) offers an increased I/O performance when accessing recent measurements and model instances from the dedicated store. The data locality is ensured by the dedicated storage manager which organizes incoming data compactly and effectively (e.g., in a required order).

Bulk-loading (traditional) offers an efficient way to populate a database managed by the analytical DBMS (see Chapter 5). Measurements and models from the dedicated store are inserted in a database in “large chunks”, which can be done much faster than inserting one tuple at a time.

Measurement-model substitution (novel) allows coping with extremely large number of updates (that might occur sporadically). Incoming measured values can be ignored completely if, within a bounded error, they follow existing prescriptions (e.g., flex-offer assignments) or invariant flexibility models (e.g., forecasted time series). If not, the new measurements can be used to update (re-forecast) invariant flexibility models while storing only new model instances or forecast model parameters – those that are used to compute these model instances. As shown in Figure 10.2, the (evolving) series of such model instances can replace the complete series of measurements, which, as we show in Section 10.4, can be exploited [123] when processing queries approximately. Since only model instances or parameters are stored, there are (in total) less data values to be retrieved from a disk and therefore the processing is much faster.

Example 10.4 (Substituting measurements with prescriptions). *Suppose that in the MIRABEL use-case a prosumer reports (factual) energy consumption measurements from each of its flexible appliances. These measurements can be ignored by an aggregator agent if the prosumer “behaves as expected” and resulting measurements precisely follow previously issued flex-offer assignments (prescriptions).*

Example 10.5 (Substituting measurements with flexibility models). *In the MIRABEL use-case, a BRP agent uses measurements from external sources to forecast inflexible consumption (and production) – instances of the invariant flexibility*

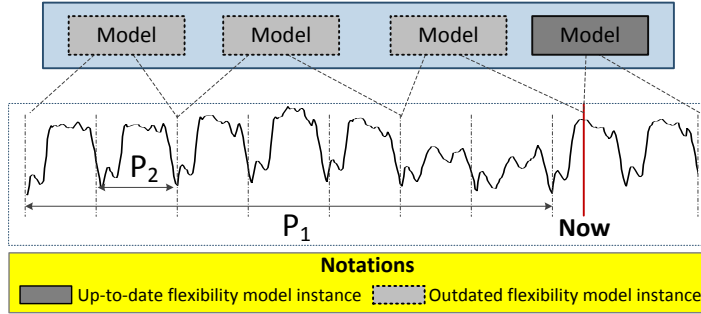


Figure 10.2: Measurements and associated invariant flexibility models

model ($ts^{cons+prod}$, see Section 3.6). These measurements can be ignored by the BRP agent if an existing instance of the invariant flexibility model describes (precisely enough) these measurements. If not, measurements can be used to update the flexibility model instance while only materializing new model instances or parameters used to compute them (and ignoring the measurements).

Model-compression (novel) may be very effective when fitting (operational) flexibility and prescription models in main memory (or a dedicated disk-based store) with limited capacity. The model-based storage of measurements (especially when forecasting parameters are stored) can also lead to (very) compact representations.

10.4 Approximate Query Processing

We use the term an *approximate query* to denote either an imperative model-specific or SQL-based declarative query that returns approximate results and can, potentially, be processed within a bounded user-specified time. An example of such *approximate query* is a declarative *solve query* relying on an iterative (black-box) solver returning approximate result after a limited number of iterations (see Chapter 8). The proposed planning component architecture allows additional types of *approximate queries* for achieving much faster (and time-bounded) processing of (1) measurements and (2) flexibility model instances, respectively.

Example 10.6 (Approximate queries in MIRABEL). *In the MIRABEL use-case, trend-analysis and long term energy planning (analytical) tolerate approximate results. Grid load balancing (operational) tolerate some imprecision in query results. However, billing and risk analysis (analytical) require very precise results. For the first two use-cases, approximate queries can be utilized to increase overall performance.*

In the use-case of **measurement** analysis, the *model-based* representation of measurements (see Section 10.3) can be utilized to speed-up the processing of queries that tolerate some amount of imprecision. Provided a maximum allowed processing time or an explicit error (absolute error or confidence) describing how large imprecisions are allowed to be, historical measurements can be reconstructed from (in-memory or materialized) instances of prescription or invariant flexibility models (as explained in Section 10.3). If these model instances cannot guarantee a required precision, actual measurement values can be retrieved (either from the dedicated store or the DBMS). An example of such approximate SQL-based query on measurements is given below.

Example 10.7 (A query for approximate measurement retrieval). *The following SQL query retrieves an energy consumption time series for the previous 24 hours with at most 5% error:*

```
SELECT RELATIVE_ERROR(0.05),
        TIME_WINDOW  (now() - INTERVAL '24_hours', now());
SELECT hour(time), SUM(energy) AS energy_per_hour
FROM GetApproximateInflexibleConsumption()
GROUP BY hour(time) ORDER BY hour(time);
```

Here, the utility functions `RELATIVE_ERROR` and `TIME_WINDOW` set the error threshold and the desired 24 hour time window, respectively. Alternatively, the utility function `MAX_TIME` specifies the maximum allowed time for a time series retrieval.

To efficiently process approximate queries involving measurements and to estimate expected costs, each model instance representing different segments of a measurement series (see Figure 10.2) can be specially annotated (providing statistics) and indexed.

In the use-case of **flexibility model** analysis, a lossy aggregation of flexibility models (the *reduce* operation) can be utilized to lower the amount of stimulus and response combinations needed to be explored, as shown in the example below.

Example 10.8 (A query for approximate nominal response calculation).

*The following model-specific approximate query defines the use of the lossy flex-offer aggregation operation (*reduce*^A from Chapter 6) to speed-up the computation of a nominal response, requiring to enumerate and to average (lots of) feasible responses.*

```
multiUnitFM      <- reduceA(getFlexOffers(), MAX_FLEXIBILITY_LOSS := 1000);
nominalResponse  <- getNominalResponse(multiUnitFM, MAX_TIME := 10);
```

Here, the bounds for the flexibility loss (`MAX_FLEXIBILITY_LOSS`) and the maximum allowed time for the nominal response calculation (`MAX_TIME`) are provided. Note, the time of computing `reduceA()` is insignificant compared to `getNominalResponse()`.

Approximate queries, as exemplified above, may increase query processing performance and allow meeting deadlines encountered in operational workloads.

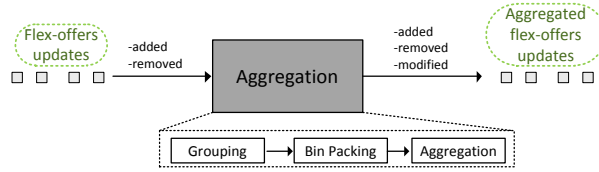


Figure 10.3: Input and output of the incremental aggregation

10.5 Subscriptions and Incremental Planning

In the continuous operation of an agent, various actions (e.g., a message exchange or a decision-making) need to be triggered when outputs of agent-specific planning operations (e.g., *reduce* or *map*) change (marginally or significantly), e.g., due to new measurements and/or flexibility and prescription model updates received from successor and predecessor agents and/or external sources.

Example 10.9 (Action triggering in MIRABEL). *In the MIRABEL use-case, an aggregator agent needs to aggregate flex-offers (using reduce^A) and then communicate the resulting aggregated flex-offers to successor BRP agents only if these aggregated flex-offers change significantly due to newly received flex-offers from predecessor agents. Similarly, a BRP agent needs to (re-)schedule flex-offers (using reduce^B) and then communicate the resulting prescription model instances to predecessor aggregator agents every time flex-offers (f^{multi}) or forecasts of inflexible consumption and production ($ts^{\text{cons+prod}}$) changes significantly.*

The traditional approach for detecting changes in planning operation (function) output requires repeatedly re-estimating the output as well as continuously polling a database for input data. This is very inefficient if instances of flexibility (or prescription) models change only marginally leading to high application costs. To prevent the polling, the proposed RT-ready planning component allows for continuous *subscription-based* queries (shown in Example 10.3), which can be registered and then efficiently processed utilizing the *subscription processor* (see Figure 10.1). Such *subscription-based* queries can be either model-specific or SQL-based, as exemplified below.

Example 10.10 (Aggregated flex-offer notifications). *The following continuous subscription-based query allows requesting aggregated flex-offers and then be notified about new aggregated flex-offers each time they change by more than 10kWh (from the previously reported flex-offers):*

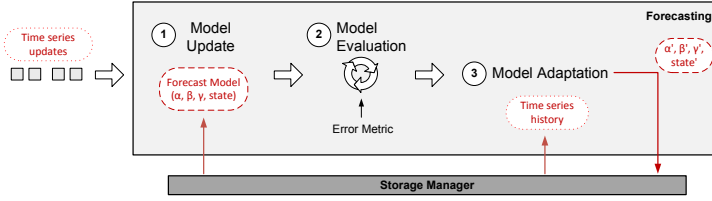


Figure 10.4: The overview of forecast model maintenance

```
NOTIFY ON energy_delta > 10000
      reduceA(getFlexOffersInDedicatedStore());
```

Notifications are caused by flex-offer inserts and deletes resulting after they are received from other agents or expire due to inherent deadlines (e.g., latest assignment time or latest start time). These flex-offer inserts and deletes can be easily processed using the proposed incremental aggregation technique (Section 6.5), allowing to update aggregated flex-offers incrementally as shown in Figure 10.3.

Example 10.11 (Forecast notifications). Similarly to the subscription-based aggregation query, the following continuous subscription-based forecast query (similar to that from Section 7.1.4) allows requesting forecasts for the next 24 hours and then be notified when they change by more than 10%:

```
NOTIFY ON error_threshold >= 0.1
SELECT hour(time), SUM(energy) energy_per_hour
FROM InflexConsumption_TSview
WHERE time IN (tomorrow())
GROUP BY hour(time)
```

To efficiently process this query, a DBMS with the forecasting capability (see Chapter 9) must be further enhanced to support the continuous maintenance of associated forecast models (see Figure 10.4). For each new measurement, a model update (1) is initiated. First, the model evaluation (2) checks a forecast model error with regard to the new measurement. If the error exceeds a given threshold, the model adaptation (3) is triggered in order to re-estimate the model parameters. In many cases, model parameters can be maintained incrementally, which is less expensive than recomputing them from scratch. A notification is generated by the DBMS only if forecasted values differ from the previously reported by more than the given threshold.

When a notification is generated, the RT-ready planning component then invokes an associated handler (e.g., from the control or the GUI component), which can also process

an updated query result incrementally for efficiency reasons, as exemplified below.

Example 10.12 (A notification-driven incremental decision-making).

When flexibility (or prescription) model instances change and associated notifications are generated, a decision-making handler is invoked for building and solving a decision model instance (solve , solve_M , or solve_R), which is costly. In many cases, an existing solution to a decision problem can be incrementally maintained using some incremental optimization technique [100] (see Section 8.6). Such techniques can, potentially, be integrated into SolveDB (see Chapter 8) such that the specifics of incremental solution maintenance are transparent to a user and encapsulated under the following data manipulation language (DML) statement:

```
REFRESH MATERIALIZED VIEW some_solve_query_view;
```

Here, $\text{some_solve_query_view}$ is a materialized view over a user-specified solve query (see Section 8.6) representing a solution to a decision problem.

In summary, subscription-based queries allow timely (not too early or late) notifications about changes of flexibility and prescription model instances, requiring no repeated requests, polling of a database, and the blocking of an executing process. The frequency of such notifications depends on a desired error, which can be freely chosen depending on a particular use-case. A use-case where subscriptions are very useful is given below.

Example 10.13 (Model propagation in the MIRABEL ICT system). *In the MIRABEL use-case, aggregator and BRP agents need to exchange flex-offers and flex-offer assignments. Instead of pulling such model instances, an agent can, potentially, register a corresponding subscription-based query and then push deltas (see Figure 10.3) when flex-offers and/or flex-offer assignments change (from the previously communicated ones) by more than a desired error, thus optimizing agent's communication costs.*

10.6 Experimental Results

We developed the initial prototype of the RT-ready planning component for the MIRABEL use-case. In the prototype, we integrated (1) a dedicated disk-based store for measurements, (2) a dedicated memory-based store for flex-offers, (3) flex-offer aggregation techniques from Chapter 6, and (4) forecasting functionality tailor-made for the energy domain [93]. We used this prototype to explore the impact of (1) the dedicated disk-based store for measurements, (2) flex-offer caching in main-memory prior-to an aggregation, and (3) different forecast model maintenance strategies. Results of the experimental evaluation are given in the corresponding sections below.

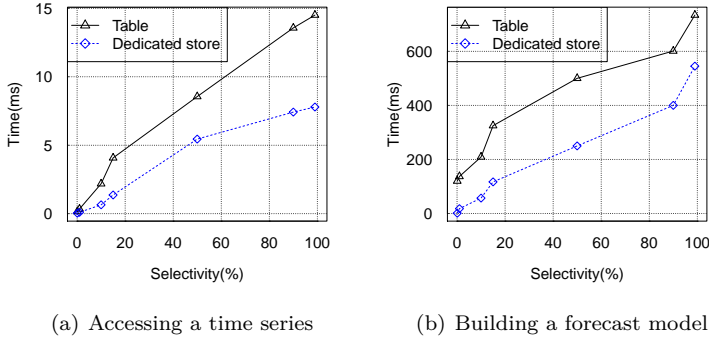


Figure 10.5: The impact of storing measurements as database table (solid line) and as a disk-block array (dotted line)

10.6.1 Experiments on dedicated storage

We now evaluate the impact of using the dedicated disk-based store for measurements (see Section 10.2). For experiments, we used a real-world time series dataset with 110,000 measurements and a computer with Intel Core i7 CPU, 8 GB of RAM, and Ubuntu (x86_64) OS. We represent the time series as (1) a database table and as (2) a native disk-block array where time series values are stored sequentially. By varying the selectivity from 0.1% to 100% of the dataset, we measure the total time of: (1) evaluating a simple query that only accesses the materialized time series data (see Figure 10.5(a)) and (2) building an instance of a forecast model which requires accessing the base data multiple times (see Figure 10.5(b)).

As seen in Figure 10.5(a), the dedicated disk-based store allows achieving 4.5 time speed-up for small time series ranges (selectivity of 0.1%) and up to 1.8 time speed-up for almost the entire dataset. This is achieved by keeping the heap organized, while only accessing the needed pages and thus minimizing the disk bandwidth. Furthermore, as no additional sorting is required, the dedicated store allows achieving up to 100 time speed-up for small time series ranges (i.e., selectivity of 0.1%) and up to 1.3 time speed-up for almost the entire dataset when building an instance of a forecast model.

10.6.2 Experiments on aggregation input caching

We now study the impact of *caching* (discussed in Section 10.3) applied prior-to the lossy flex-offer aggregation (*reduce^A*). In this experiment, we evaluate the throughput of flex-offer updates that can be processed with our proposed single-time and incremental flex-offer aggregation techniques from Chapter 6 when different levels of caching are

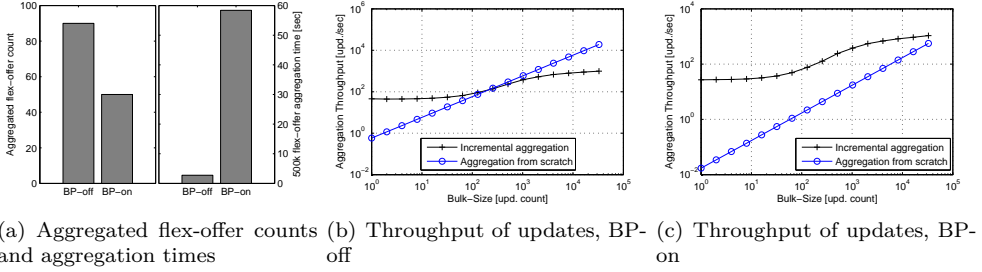


Figure 10.6: The impact of caching flex-offer aggregation input

employed. For the experiment, we use exactly the same experimental setup and the dataset as in Section 6.6.2 and set aggregation parameters so that the best scheduling result is obtained ($TFT = 0$ and $EST = 7$). Like in the experiment from Section 6.6.1, we initially aggregate 500000 flex-offers. Then, we continuously feed new and remove existing flex-offers while keeping the total number of micro flex-offers equal to 500000. In contrast to the experiment from Section 6.6.1, here we first cache (store) such flex-offer inserts and deletes in the dedicated main-memory store and then process them in bulks, where the size of a bulk determines the level of aggregation result up-to-dateness, i.e., small bulks yield up-to-date results while larger bulks yield slightly outdated results. When the bulk of a certain size is formed, we trigger the aggregation. We evaluate throughputs when the bin-packing (BP) feature is on and off. The BP-on means that macro flex-offers are guaranteed to have the time flexibility of no less than 2 hours.

Figure 10.6(a) shows the 500000 flex-offer aggregation performance. When the BP is off, 500000 micro flex-offers are aggregated into 90 macro flex-offers. When the BP is on, the number of macro flex-offer is only 50 as some of the micro flex-offers are excluded since they result in macro flex-offers with time flexibility lower than 2 hours (which is not allowed by the aggregate constraint). Times spent aggregating with BP-off and BP-on are 3 and 58 seconds, respectively. Figure 10.6(b) shows throughputs of updates when the BP is off and (1) aggregating 500000 flex-offers from scratch for each a new bulk (line with circles) and (2) incrementally maintaining macro flex-offers using bulks of flex-offers (line with crosses). The throughputs of updates when the BP is on are shown in Figure 10.6(c). As seen in the figures, the caching allows to speed-up the aggregation in all cases (single-time/incremental, BP-on/off), but always results in (slightly) outdated results, where the level of result up-to-dateness is determined by the size of a bulk (see Figures 10.6(b)–10.6(c)). Caching is more effective when flex-offers are aggregated from scratch. When bulks are sufficiently large, caching may allow achieving a throughput that is comparable to that of the incremental aggregation. Note that when no caching is used, as seen in the figures, the throughput of the incremental aggregation is by two orders of magnitude higher compared to the single-time aggregation.

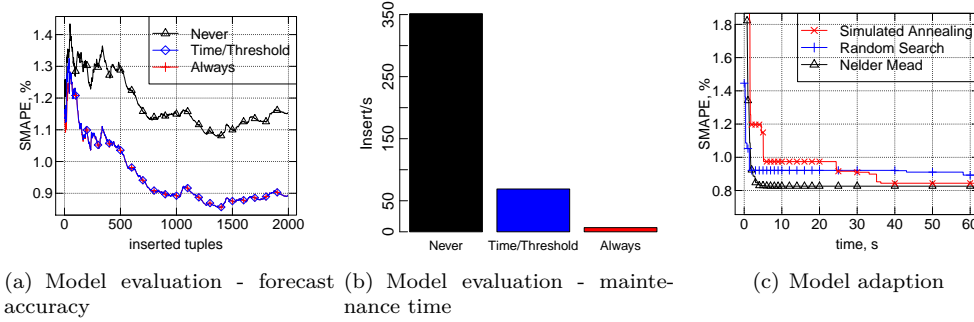


Figure 10.7: The influence of maintenance strategies and model adaption performance

10.6.3 Experiments on forecast model maintenance

We now evaluate the influence of different forecast model evaluation and adaption strategies (for the use-case from Example 10.11) using a publicly available energy production data set [124]. It consists of a single aggregated wind energy time series from 2004 to 2006 in 10 min resolution. The experiment is conducted using Holt-Winters Triple Seasonal Exponential Smoothing, which is a forecast model tailor-made for the energy domain [93]. To measure the forecast accuracy, we use the symmetric mean absolute percentage error (SMAPE), which is a scale-independent accuracy measure and takes values between 0% and 100%. The following experiments were executed on an IBM Blade (Suse Linux, 64 bit) with two processors and 4 GB RAM.

In a first experiment, we explore the influence of different model evaluation strategies. A first model evaluation strategy *never* adapts the model parameters, while a second strategy *always* adapts the forecast model after each new measurement. Finally, our third strategy triggers a model adaption based on a combined time- and threshold-based approach. For this last approach, we set the time limit to 24 hours and the threshold to 2.5%. Figure 10.7(a) shows the development of the model accuracy for a series of inserts on the time series for each of the three strategies, while Figure 10.7(b) shows the estimated number of inserts per second we can achieve with the corresponding strategy. First of all, we can observe that model adaption always leads to a lower forecast error compared to no adaption at all and thus is required to achieve the best possible accuracy. However, if we adapt the model after each new measurement we achieve a much lower throughput compared to no adaption. In comparison, our time- and threshold-based approach still leads to a low forecast error but increases the throughput.

In a second experiment, we examine a single model adaption step more closely by comparing the error development of three important parameter estimation strategies (Figure 10.7(c)). In detail, we measure the accuracy over time of a single adaption step

using Simulated Annealing, Random Search, and Random Restart Nelder Mead. As it can be seen, all algorithms converge to a result having similar accuracy, where Random Restart Nelder Mead requires much less time than, for example, Simulated Annealing. Thus, the model adaption strategy has a high impact on the required time and needs to be set accordingly.

10.7 Summary and Discussion

We have presented the architecture of the agent's planning component (Section 4.1) for the real-time setting. The architecture enables *storage* and *query processing* optimizations and offers many useful features such as *model-specific*, *approximate*, and *subscription-based* queries. Experiments with our initial MIRABEL-specific prototype, integrating (some of) the proposed techniques, witness significant performance improvements when storing/accessing (historical) measurements, aggregating flexibility models, and maintaining forecast models.

In order use the proposed optimizations and features when designing other PRESCRIPTIVECPS instances for the real-time setting, tailor-made customizations and adaptations have to be made in order to adapt to and to exploit properties and specifics of flexibility and prescription models employed by a concrete PRESCRIPTIVECPS instance. Note, tailor-made customizations and adaptations, in contrast, are not required by the generic DBMS with built-in analytics (Chapters 7–9).

When such a tailor-made RT-ready planning component is combined with the generic analytical DBMS (Chapters 4–10) and carefully integrated into a single joint system of an agent, the agent is equipped with a very powerful prescriptive analytics technology allowing to meet all the stated flexibility and prescription model management requirements (Requirements 1–10), which is not possible with existing systems. We now briefly contrast the capabilities of such integrated agent software system to existing work.

Event engines [125], for example, do not support complex analytical queries and advanced analytics (e.g., *forecasting* or *optimization problem solving*). Pure data stream management systems [126] can cope with high-rate input streams and rather complex queries, however, ad-hoc data analysis and advanced analytics on historical, present, or future data are impossible. From the more traditional DBMS side, read-optimized data organization such as column-oriented data management or hybrid OLTP/OLAP solutions [127] provide efficient ad-hoc data analysis but exhibit drawbacks with regard to write-intensive applications. In the traditional (descriptive) data warehouse area, real-time data warehouse approaches try to cope with a continuous stream of write-only updates and read-only queries at the same time [128]. However, they still lack many prescriptive analytics tools (e.g., *forecasting* or *optimization problem solving*) and none of them natively support flexibility, decision, and prescription models.

Chapter 11

Flexibility and Prescription Model Visualization

In this chapter, we focus on the graphical user interface (GUI) component of a PRESCRIPTIVECPS agent software system and show how flexibility and prescription models can be visualized and graphically analysed. In general case, the required model analysis and visualisation is specific to a PRESCRIPTIVECPS agent and/or agent role, and depends on the employed types of flexibility and prescription models, users, business processes, etc. For specificity, we focus on the MIRABEL use-case and aim to design and implement a *visual model analysis framework*, considering agents in the *BRP* role and their used *BG* flexibility and prescription models (see Section 3.6). First, in Section 11.1, we formulate high-level requirements for such a visualization framework, considering the MIRABEL use-case. Then, in Section 11.2, we present our initial results implementing visual model analysis framework for the *BRP* role. Finally, in Sections 11.3–11.4, we discuss related work and generalize the proposed model visualization ideas for other instances of PRESCRIPTIVECPS. This chapter is based on Publication [9].

11.1 Model Visualization and Analysis in MIRABEL

Consider the MIRABEL use-case and a potential BRP enterprise using the discussed PRESCRIPTIVECPS agent software system (Chapter 4). The BRP’s system (potentially) integrates the most of the analytical capabilities, compared to corresponding systems of other MIRABEL actors. As shown earlier, such agent system manages *BG* flexibility and prescription model instances represented as sets of flex-offers, flex-offer assignments, and time series (Section 3.6). To perform online and offline analysis of all such data, (human) users require model visualization and visual analysis tools, e.g., for decision

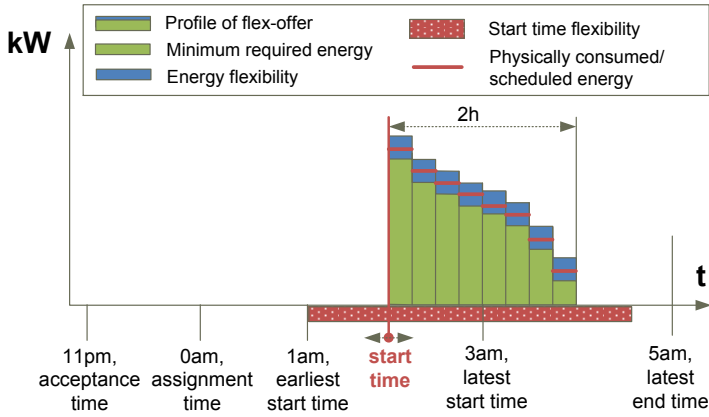


Figure 11.1: The structural elements of flex-offers/flex-offer assignments

models building and adaptation (Requirement 9) and other use-cases.

Example 11.1 (Flexibility and prescription model analysis). *To find out the reason behind the shortage of electricity, it is important for BRP to be able to see relevant geographical areas on a map with an option to drill down to the level of individual flex-offers.*

We now consider such a BRP enterprise and present the essential types of views over flex-offer data in the form of requirements for a BRP-specific graphical user interface (GUI) – referred as *visual model analysis framework*. We assume flex-offers with the complete (rich and realistic) set of attributes discussed in Chapter 5.

Given flexibility and prescription model (*BG*) instances stored in a database (Chapter 5), the framework must allow for an in-depth analysis of individual model parts (time series and flex-offers) and summaries over models as the whole (e.g., the amount of stimulus). When visualizing individual model parts, the discussed elements of a flex-offer and a flex-offer assignment (shown in Figure 11.1) must be clearly visible. For time series data, traditional time series visualization techniques (to be surveying in Section 11.3) must be provided. When visualizing model summaries, the following statistics, among others, are essential and must be provided:

Flex-offer count defining the total number of accepted, assigned, or rejected flex-offers.



Figure 11.2: The example of the map view of flex-offers

Flex-offer attribute value summarizing the values of a particular flex-offer attribute, e.g., minimum/maximum/average energy or time flexibility defined by flex-offers.

Scheduled energy defining the amount of energy that is planned utilizing flex-offers.

Amount of stimulus and response providing an estimate on how well energy can be balanced utilizing flex-offers (see Section 6.2).

Forecast/plan deviations specifying the difference between the amounts of forecasted/scheduled energy and actually (physically) consumed or produced energy, considering that the flexibility model captures the (invariant) measurements of actually consumed amount of energy.

For all flex-offer elements and statistics, the framework must support *filtering* and *grouping* on the following types of flex-offer attributes:

Temporal allowing to select data for a particular time interval and to analyse data at different time granularities.

Spatial geographical allowing to select data for (or group on) a spatial object, e.g., country, city, or district. A user-friendly view (see an example in Figure 11.2) to explore and filter flex-offer data on a map must be provided.

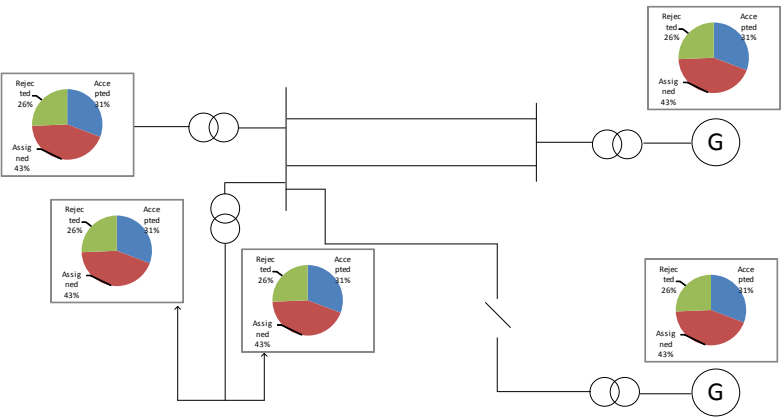


Figure 11.3: The example of the schematic view of flex-offers

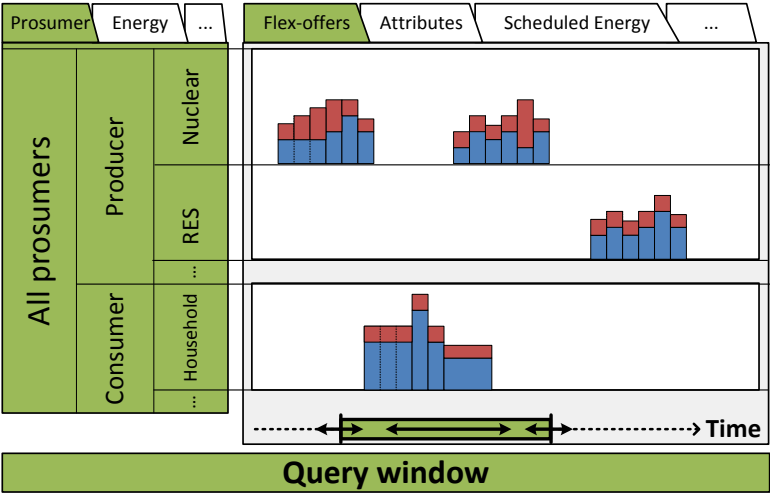


Figure 11.4: The example of the pivot view of flex-offers

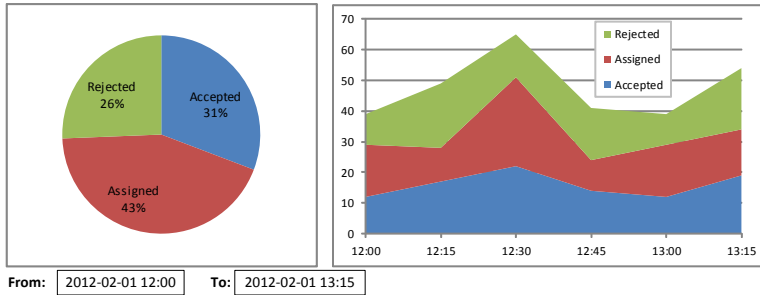


Figure 11.5: The example of the dashboard view of flex-offers

Spatial topological allowing to select data for (or group on) the topological or electrical structure the electricity grid, e.g., for a particular 110kV transmission line. A user-friendly view (see an example in Figure 11.3) to explore and filter flex-offer data on a topological map must be provided.

Energy type allowing to select data associated with a particular energy type, e.g., renewable energy from hydro power plants.

Prosumer type allowing to select data associated with a particular prosumer type, e.g., small industrial power plants.

Appliance type allowing to select data associated with a particular appliance type, e.g., electric vehicles.

In addition, the nesting of multiple filtering and grouping operations must be supported.

Example 11.2 (Multiple grouping attributes). *Counts of accepted flex-offers in the west Denmark in the period from Jan-2013 to Feb-2013 grouped by cities and energy type are interesting and must be retrieved by the framework.*

Moreover, to support filtering and grouping tasks, intuitive dimension hierarchies as those in OLAP has to be created for all these types of attributes. A convenient OLAP-based “pivot-table” view to explore flex-offer data must be provided (see an example in Figure 11.4). In the view, a user must be able to choose a preferred dimension hierarchy (e.g., prosumer type), navigate (drill up and down) hierarchy members from the most summarized (e.g., all prosumers) to the most detailed (e.g., household), and analyse the preferred elements or the measures (e.g., flex-offers, see above for more) on

multiple swimlanes in the view. For the view, a possibility to manually formulate a query (e.g., SQL or *model-specific*, as shown in Chapter 10) must be provided. Finally, a view to summarize the complete flex-offer data for the selected time interval (see an example in Figure 11.5) must be offered.

In the next section, we describe our initial results when implementing the BRP-specific visual model analysis framework.

11.2 Initial Results When Implementing Visual Model Analysis Framework

We have built a visualization tool offering only the subset of the described features (defined in Section 11.1). The tool is capable of visualizing a large number of flex-offers on a computer screen and it allows interactively performing an in-depth analysis of individual flex-offers. It currently supports the flex-offer aggregation, the disaggregation, and the basic filtering functionality. Time series are visualized as flex-offers with no flexibility (stimulus). In the rest of the section, we provide a walk-through of its functionality by starting with the data loading and ending with its useful graphical enhancements.

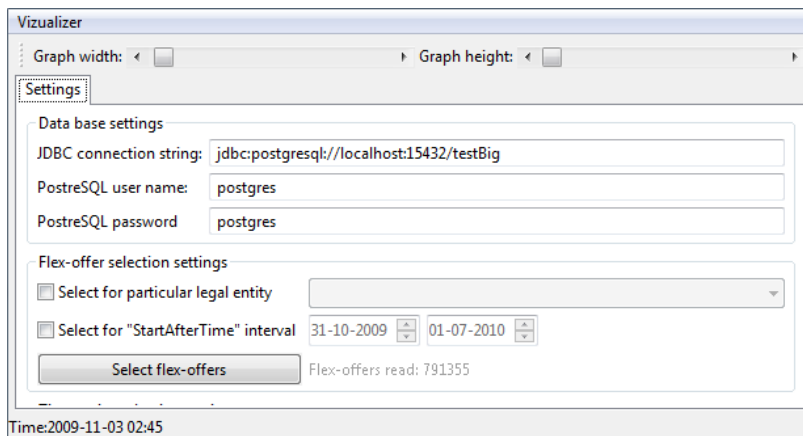


Figure 11.6: The flex-offer loading tab in the main window

First, the tool reads flex-offers and related data from a database employing the MIRABEL DW schema, presented in Chapter 5. When the tool connects to a database (PostgreSQL), a user can choose a specific legal entity (a prosumer), whose flex-offers the user wishes to visualize (see Figure 11.6). The user can also select an absolute time interval, for which flex-offers needs to be selected.

When flex-offers are read, a new *flex-offer view* tab is created in the main application

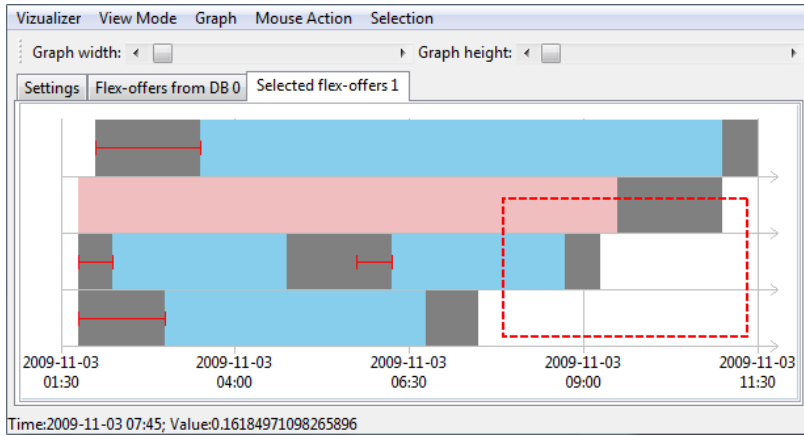


Figure 11.7: The basic view of flex-offers

window (see additional tabs for two read operations in Figure 11.7). There are two flex-offer views currently supported: the basic and the profile view. Both these views show the set of flex-offers in a graph with the abscissa axis corresponding to time. Depending on the view, the ordinate axis in the graph is unit-less (see Figure 11.7) or shows energy (see Figure 11.8). As flex-offers are temporal object which may potentially overlap in time, boxes representing flex-offers (see Figures 11.7–11.8) are stacked on each other thus occupying one of several ordinate axes in the graph.

The basic view (see Figure 11.7) is used to show a large numbers of flex-offers by visualizing only the most essential properties of a flex-offer: (1) duration of energy profile (light blue or red rectangles), (2) time flexibility interval (grey rectangles); (3) scheduled starting time of a respective appliance (red solid lines). Aggregated (light red rectangles) and non-aggregated (light blue rectangles) flex-offers are distinguished by a colour. Such flex-offer view allows observing abnormalities in both individual flex-offers (e.g., unexpectedly long energy profiles) and large flex-offer sets (e.g., missing assignments in some time interval).

The profile view (see Figure 11.8) is used to provide the detailed flex-offer representation (see Section 11.1). The view is effective for a smaller flex-offer set with less than few thousands of flex-offers. It allows exploring the actual (minimum and maximum) energy bounds at every profile interval (slice), and, thanks to the synchronized scales of all ordinate axes, comparing them across multiple flex-offers. In addition to energy bounds, it shows the scheduled amounts for every profile interval (red solid lines).

Irrespectively for the selected view, the visualization tool provides additional information about flex-offers when pointing their representations with a mouse pointer (see Figure 11.9). This includes the markers (yellow lines) for user-specified *creation/ac-*

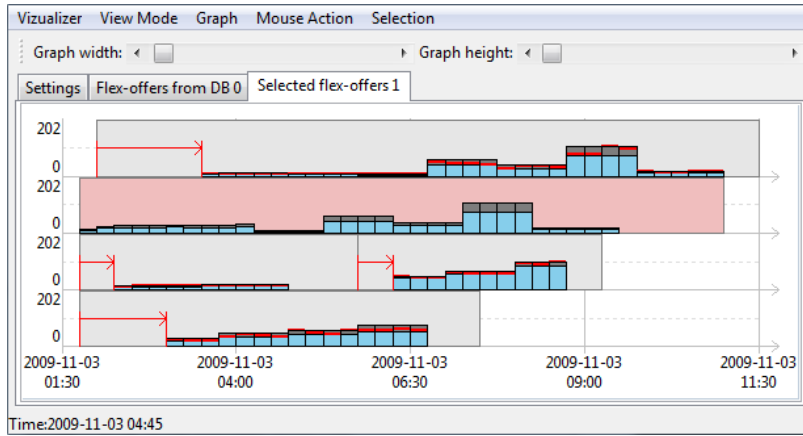


Figure 11.8: The profile view of flex-offers

ceptance/assignment times of a flex-offer as well as indications (red dashed lines) on which flex-offers were aggregated to produce the pointed flex-offer. The mouse action can be changed to allow the interactive selection of flex-offers. Flex-offers can be selected one-by-one or by drawing a rectangle (see dashed red rectangle in Figure 11.7). The selected flex-offers can be shown on a different tab, removed from the current view, or processed with the tools from the main menu.

The visualization tool integrates the flex-offer aggregation and disaggregation functionalities, presented in Chapter 6. This allows reducing the count of flex-offers shown on a screen by aggregation, and interactively tuning values of the aggregation parameters.

Finally, the tool offers useful graphical enhancements such as automatic selection of “pretty scales” of the axes, as well as the incremental rendering of flex-offers, which allows executing actions when a flex-offer rendering is in progress.

As mentioned before, the presented flex-offer visualization tool offers only the subset of the functionality of the complete the visual flex-offer analysis framework. As the next immediate enhancement, the basic and the detailed views will be integrated into the pivot view (see the description of Figure 11.4), where the flex-offer aggregation will be applied to produce inputs for the flex-offer visualization on swimlanes (see Figure 11.4). Later, the remaining functionality of the framework will be implemented.

In the next section, the related work is reviewed.

11.3 Related Work

The information visualization is an active and still emerging field offering approaches to see, explore, and understand large amounts of information in intuitive ways. Con-

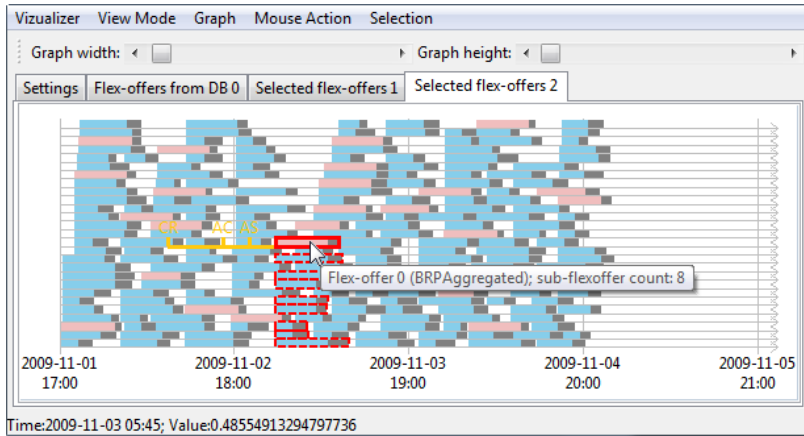


Figure 11.9: On-the-fly information about flex-offers

sequently, much work has been done in this broad discipline, and there is a number of publications surveying the accomplished results [129–131] and guides on how one should properly display [132] and visually analyse [133] information. In this context, flex-offers being special instances of flexibility models are novel concepts and thus the visualization of flex-offers has not been addressed yet. Nevertheless, flex-offers can also be seen as instances of multi-dimensional, spatial, temporal, spatio-temporal, and energy data and methods for effectively visualizing such type of data has already been proposed.

First, the well-known OLAP (Online Analytical Processing) techniques to manage multidimensional data led to the research field of Visual OLAP [134]. There, the traditional 2D interface for analysing multidimensional data is a pivot table (or cross-tab) view [135]. The current state-of-the-art (the survey [135] is available) enhances the pivot table view via providing the set of popular visualization techniques, such as bar-charts, pie-charts, and time series, as well as more sophisticated visualization layouts such as scatter plots, maps, tree-maps, cartograms, matrices, grids, and vendors's proprietary visualizations (e.g., decomposition trees and fractal maps). Recently, 3D interfaces for OLAP were explored [136].

Second, more general approaches to visualize multidimensional data exist [137, 138]. According to the survey [139], existing multidimensional visualization techniques are classified as pixel-oriented (map data value to a coloured pixel), geometric projection (e.g., parallel coordinates), iconography (map each data item to an icon), and hierarchical display techniques (e.g., dimensional stacking, tree-maps).

In addition, a variety of methods for visualizing time-oriented data have been proposed. The survey [140] and the categorization [141] (systematic view) on the diversity of all these methods exist. According to the categorization, methods are classified ac-

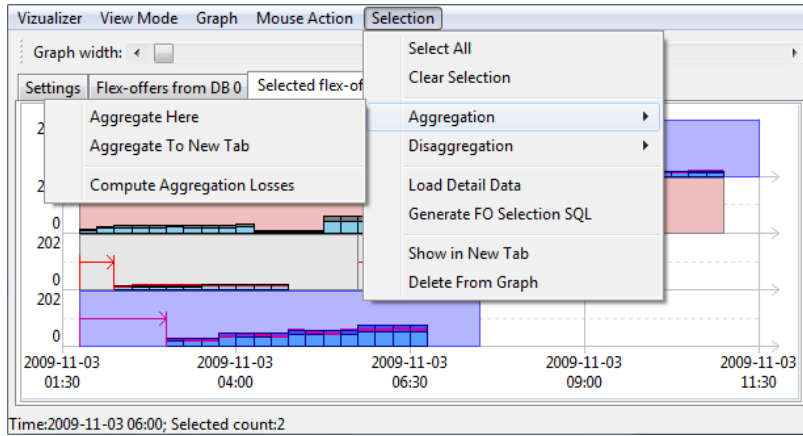


Figure 11.10: Available tools for interactively managing flex-offers

cording to the (1) characteristics of time axis (e.g., time points v.s. intervals), (2) data type (e.g., univariate v.s. multivariate), and (3) data representation (e.g., still images v.s. animations). Time-series is the special case of a time-oriented data, and a number of visualization techniques were proposed [142–144].

For spatial data, a number of cartography/geo-visualization [145], topology-based [146], and domain-specific [147] techniques were developed. In addition, techniques to visualize spatio-temporal data [148] are classified according to (1) the types of spatio-temporal data they are applicable to, and (2) the exploratory tasks they can potentially support. Finally, techniques to visualize data of the regular [149, 150] as well as of the smart [151] electricity grid were proposed, but those - do not consider the visualization of flex-offer.

In the context of the surveyed work, our implemented flex-offer visualization tool employs the variation of histograms plot based on the dimensional stacking method [138], where 2-dimensional subspaces are stacked onto each other. For the complete visual model analysis framework, existing visualization techniques, particularly those for OLAP, geo-visualization, time series, and electricity data will have to carefully adapted and integrated to support comprehensive visualization and analysis of flex-offers.

11.4 Summary and Discussion

In this chapter, we elaborated the graphical user interface (GUI) component of a PRESCRIPTIVECPS agent software system using the MIRABEL use-case and considering an agent in the *BRP* role. First, we argued that the visualization of flexibility and prescription models is important and is required by (human) users acting on behalf of a BRP, for example, for a decision model building and adaptation (Requirement 9). Then, we

have formulated high level requirements for BRP-specific visual model analysis framework, which supports various visual analytic activities within the MIRABEL enterprise. Such framework is required to provide (1) detail and summarized views over instances of flexibility models (flex-offers) and (2) convenient OLAP-inspired approaches to navigate and explore model data. Then, we have presented the flex-offer visualization tool as our initial result implementing the complete framework. The flex-offer visualization tool meets only the subset of all requirements for the framework, but it offers a novel approach to visualize a large number of individual flex-offers. The approach employs the variation of the histogram plot where 2-dimensional (time and energy) subspaces are stacked onto each other. We have also discussed the immediate extensions of the tool as milestones developing the full-featured visual model analysis framework. Finally, as flex-offers can be seen are instances of multi-dimensional, spatial, temporal, spatio-temporal, and energy data, we have reviewed the related work on the visualization of those particular types of data.

In other instances of PRESCRIPTIVECPS, flexibility and prescription models may, potentially, be complex multi-dimensional entities exhibiting temporal, spatial, spatio-temporal characteristics. For visualization of these, we encourage the use of the visualization techniques introduced and surveyed in this chapter, of course, customized and adapted accordingly to suite a particular use-case, agent role, or an individual agent.

The future work developing the BRP-specific visual model analysis framework encompasses the implementation of all the remaining functionality of the visualization framework with a sufficient quantitative evaluation on how well the functionality serves the end-user needs. Furthermore, an interesting future work is the development of the integrated energy planning and control platform offering high level qualitative information such as alerts about expected shortages or over-capacities and an option to drill down data to find out a reason behind this. The platform might couple the functionalities of the existing SCADA, ERP, planning, and auction bidding systems, and, in addition to flex-offers, might integrate other types of data such as costs, prices, energy measurements and constraints as well as the forecasts of energy, weather, and the flexibility of different types.

Chapter 12

Conclusion and Future Research Directions

This thesis addressed the problem of (1) designing a CPS for mixed-initiative (human and machine-based) distributed continuous planning, (2) applying such a design in a concrete real-world use-case, and (3) developing a software that supports the planning and control activities and offers (human) users *prescriptive analytics* tools. While addressing this problem, the thesis presented a so-called PRESCRIPTIVECPS – which is (the conceptual model of) a multi-agent, multi-role, and multi-level CPS with distributed continuous planning capabilities. PRESCRIPTIVECPS was demonstrated in the use-case of the MIRABEL project, where a large-scale CPS, shown to be a specialized instance of PRESCRIPTIVECPS, was designed and prototyped for balancing electricity consumption and production in the power grid with a large number of renewable energy sources (e.g., windmills). The thesis proposed and demonstrated software that can (potentially) be used by PRESCRIPTIVECPS agents to support their pursued planning and control activities. The proposed agent software integrates many data management and prescriptive analytics techniques (incl., *model aggregation/disaggregation*, *problem solving*, *real-time processing*, and *visualisation* techniques), which were demonstrated in the thesis, together with the results of their experimental evaluation. PRESCRIPTIVECPS was described at the three levels of scale: (1) the level of system of agents (CPS), (2) the level of an agent, and (3) the level an agent software component. Contributions at all these levels are summarized in the individual parts of the next sections.

12.1 Contributions

We now overview the contributions of this thesis by using the notations and elaborating on (where relevant) the initial summaries of contributions from Section 1.6.

Contributions at the CPS level

1. In Section 3.1, the definition of PRESCRIPTIVECPS was provided. PRESCRIPTIVECPS was defined as the system of interacting *physical* and *cyber* (sub-)systems, where the cyber system is a system of systems, consisting of a set of inter-connected agents that form a hierarchy based on their obedience to each other. Agents collectively manage instances of so-called *flexibility* (world), *decision*, and *prescription* models, which are short-lived, focus on the future, and represent (1) capability, an (2) (agent owner) intention, and (3) actions to change the behaviour (state) of a physical system, respectively.
2. In Section 3.2, the definition of *flexibility*, *decision*, and *prescription* models was presented, by showing how these models are related and formally defined.
3. In Section 3.3, the semantics of PRESCRIPTIVECPS was given in terms of core planning operations (e.g., *map*, *reduce*, *solve*), which transform *flexibility*, *decision*, and *prescription* model instances.
4. In Section 3.5, the set of agent roles were defined. An agent can, potentially, take one or more of such roles depending on the set of core planning operations it supports.
5. In Section 3.6, the defined concepts were elaborated using the specialized PRESCRIPTIVECPS instance from the MIRABEL project.




Contributions at the agent level

6. In Section 3.7, data management functional and non-functional requirements were specified separately for all PRESCRIPTIVECPS agents and specific agent roles.
7. In Section 4.1, the architecture of an agent software system was proposed. The architecture integrates the number of components specially designed or enhanced to satisfy the agent-specific functional and non-functional requirements. Additionally, the overviews of capabilities and functionality of these components were provided.

Contributions at the agent component level

8. ■ In Chapter 5, the schema of a *multi-dimensional data warehouse* for the MIRABEL use-case was proposed and generalized for other PRESCRIPTIVECPS instances. The proposed MIRABEL data warehouse schema can be used for storing MIRABEL-specific instances of flexibility and prescription models (and related data). It has a number of interesting complexities such as facts about facts and composed non-atomic facts. Alternatives schemas using denormalization and arrays were considered and experimentally evaluated, but based on the performance and space usage, the chosen design is favourable. Other similar instances of PRESCRIPTIVECPS may reuse the modelling strategies and/or schema elements applied for the MIRABEL use-case, e.g., the modelling of *roles* and *time series*.
9. ■ In Chapter 6, the techniques to *aggregate* flexibility model instances and *disaggregate* prescription model instances were demonstrated in the MIRABEL use-case and generalized for other PRESCRIPTIVECPS instances. Utilizing extensive experiments, it was shown that an efficient (lossy) aggregation allows (substantially) reducing the complexity of decision problems and thus obtaining (up to 20 times) better quality decisions (solutions) in cases when solving (decision-making) time is limited. An efficient incremental aggregation technique was presented, allowing to update aggregated model instances incrementally (after changes in input) and requiring no complete re-computation.
10. ☒ In Section 7.1, the design of a DBMS with built-in time series *forecasting* functionality was presented. The DBMS is based on the standard ANSI/SPARC-based architecture and supports so-called *forecast queries* allowing to compute forecasted (future) time series values (to be used as part of flexibility models) based on historical measurements in a database. The DBMS allows transparent, efficient, and end-to-end execution of *forecast queries* and offers increased tool usability, user productivity, and forecasting performance compared to the traditional forecasting process. Various integration aspects, requirements, and challenges associated to built-in *time series forecasting* were discussed.
11. ☒ In Section 7.2, the domain-specific (MIRABEL-specific) software architecture and the approaches for *forecasting* instances of flexibility models were presented. The demonstrated approaches utilize detailed information about a domain to convert (augment) forecasted time series into flexibility model instances specific to the MIRABEL use-case.
12. ■ In Chapter 8, the design of a DBMS with built-in *optimization* problem solving functionality was presented. The DBMS enables so-called *solve queries* that offer a common language for queries and optimization models, both expressed using Structural Query Language (SQL) constructs. As part the contribution, the extensible solver infrastructure was presented. It allows integrating (into a DBMS) a

variety of solvers, each of which tackles a specific class of problems, e.g., *linear programming*. Additionally, query optimization techniques were presented, allowing to increase the execution performance and/or result quality. The presented results of extensive experiments with the PostgreSQL-based implementation showed that the integrated DBMS significantly reduces the number of lines of code (by 1.3-15 times), I/O time (by 1.7-18.6 times), and total time (up to 62 times with optimizations applied) compared with existing optimization software, leading to increased user productivity and problem solving performance.

13.  In Chapter 9, a what-if analysis based on hypothetical *what-if scenarios* and flexibility and prescription models was demonstrated in the MIRABEL use-case. For the demonstration, we proposed the extension of SQL for being able to specify scenarios and to process what-if queries based on these scenarios inside an agent's DBMS. Utilizing built-in *forecasting* and *optimization* functionalities as well as what-if scenarios and flexibility and prescription models stored in a database, an analyst is offered simplified powerful ways to analyse past, current, and future data.
14.  In Chapter 10, the real-time data management architecture for processing instances of flexibility and prescription models under (soft or hard) timing constraints was proposed. The proposed architecture enables *storage* and *query processing optimizations* and offers many useful features such as *model-specific*, *approximate*, and *subscription-based* queries. The experiments with our initial MIRABEL-specific prototype, integrating (some of) the proposed techniques, witnessed significant performance improvements when storing/accessing (historical) measurements, aggregating flexibility models, and maintaining forecast models.
15.  In Chapter 11, the design of the MIRABEL-specific *graphical user interface* (GUI) was presented. It allows displaying and visually analysing instances of flexibility and prescription models that are specific to a particular agent role (*BRP*) in the MIRABEL use-case. The presented initial implementation is able to visualize large amounts of such complex model instances and offer generalized and in-depth analysis of all such data.

12.2 Discussion

Functionality underlying all these contributions is the integral part of our proposed PRESCRIPTIVECPS. Unlike CPSs existing today, PRESCRIPTIVECPS continuously, automatically, an in (near) real-time (1) collects and consolidates information about the physical world, (2) makes predictions, (3) solves optimisation problem instances to find (optimal) decisions, and (4) finally applies these decisions to the physical world. It was shown that multiple instances of these planning activities may coexist, and they

are performed in parallel (in the distributed fashion) by, potentially, a large number of PRESCRIPTIVECPS agents in different roles such as *sensor*, *actuator*, *forecasting*, *aggregator*, *disaggregator*, and *global prescriptor*, with the activity delegations shown in Figure 12.1.

PRESCRIPTIVECPS can automatically take decisions, but, unlike artificial intelligence (AI) systems, lacks the ability of setting and achieving goals autonomously without human intervention. For setting goals, and for monitoring that the goals are actually met, inputs from (human) analysts are required (see Figure 12.1). PRESCRIPTIVECPS allows setting goals for each agent independently, and these goals might be complementary or conflicting, resulting into, potentially, complex intermixed collaborative and competitive patterns of agents' behaviour. For analysing single agent behaviour under various goals set, the thesis described various software-based *prescriptive analytics* tools helping (human) analysts to ask various complex questions about past, presence, and future. However, they were not designed for assessing the joint behaviour of multiple (or all) PRESCRIPTIVECPS agents. For this, additional PRESCRIPTIVECPS simulation models (environments) are required, as pointed by future work in the next section.

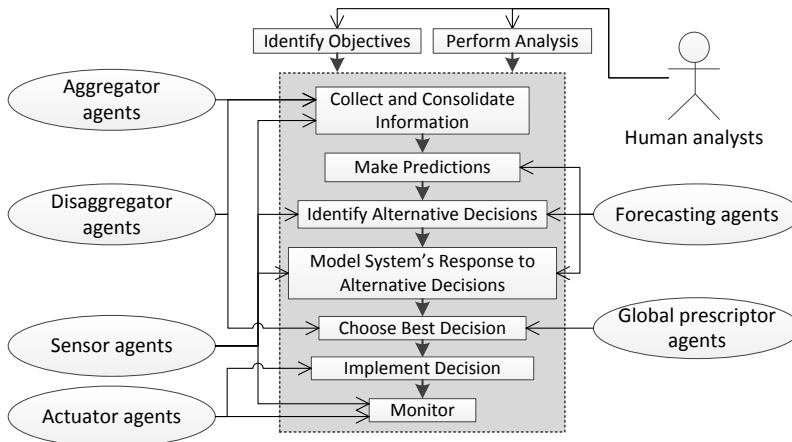


Figure 12.1: The involvement of PRESCRIPTIVECPS roles and (human) analysts in different activities of planning and control

We believe that this thesis makes a significant step towards developing technology for CPSs to be witnessed in the future. In contrast to traditional CPSs existing today, our proposed PRESCRIPTIVECPS, with all the contributions of this thesis, allows accommodating all of the following complexities, often raising insurmountable challenges for the traditional systems (see Figure 12.2):

- Large numbers of agents (e.g., for each household in Europe, considering the

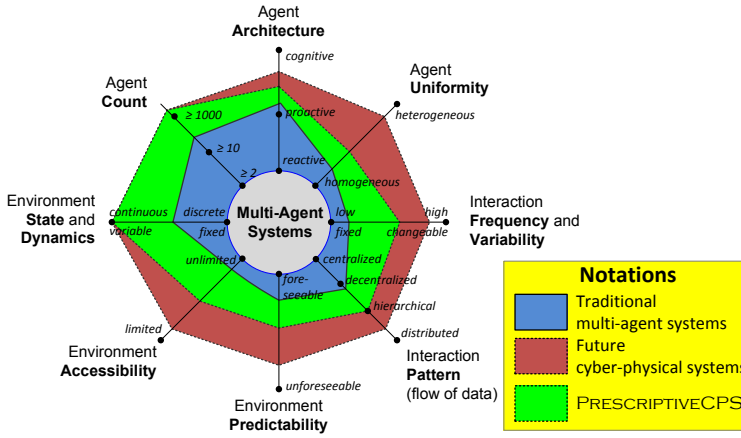


Figure 12.2: The capabilities of PRESCRIPTIVECPS in the context of traditional and future CPSs

MIRABEL use-case) which are semi-homogeneous (i.e., differing in terms of their roles) and with substantial cognitive capabilities (incl., *forecasting*, *optimization*, and *what-if*).

- Volatile continuous physical systems (environments) where accurate long-term predictions are not feasible or possible only higher aggregation levels.
- Hierarchical agent organization patterns which reflect the view of the physical world (e.g., *prosumer*, *BRP*, and *TSO*).

To counter these complexities, the theories and techniques presented in this thesis allow developing large-scale (e.g., continent- or world-scale) CPSs, which asynchronously, continuously, and in (near) real-time transform (drive) momentary knowledge about the physical world into prescriptions (decisions) – where the physical world is perceived and actuated by a large number of *sensor* and *actuator* agents, respectively. This transformation is done via *decision-making* agents that not only generate prescriptions but also *fuse*, *consolidate*, and *reshape* a knowledge about the physical world, and pass the knowledge to other (higher-level) *decision-making* agents if required. To counter (and take advantages of) sporadic and rapid changes in a volatile physical system, the proposed real-time optimisations and incremental techniques can be utilized. Alternatively, decision-making can be brought “close to” the physical system by configuring a single agent (or adjacent peer agents) to take the *sensor*, *actuator*, and *decision-making* roles and thus lowering a decision making latency. If needed, agents can be organized to mirror the organizational view of the physical world.

Furthermore, this thesis proposed software-based techniques allowing to equip agents with substantial cognitive capabilities (when needed). The most significant are the novel in-DBMS techniques allowing to (1) forecast, (2) solve optimisation problems, and (3) perform what-if analysis “close-to-the-data”, transparently to the user, and using declarative SQL-based constructs. None of such integrated data management and analytics functionalities (individually or as a group) were previously considered in the literature and/or are supported by business intelligence (BI) tools existing today.

In spite of all the thesis contributions, there are still many challenges to be addressed for future CPSs (see Figure 12.2). Some of these challenges have not been considered in this thesis but can be tackled by specialized PRESCRIPTIVECPS instances. Remaining challenges are insurmountable for PRESCRIPTIVECPS. While following the limitations shown in Figure 12.2, challenges from both categories are exemplified below.

PRESCRIPTIVECPS may support, but the thesis did not consider agents that are (1) *intelligent* (cognitive) and are able to set and achieve goals autonomously and learn from past experience, (2) *very heterogeneous* ranging from nano-scale resource-constrained devices to powerful large-scale cloud-based systems, and/or (3) *with limited access* to the physical environment that is hardly *foreseeable*.

PRESCRIPTIVECPS is susceptible to a *single point of failure* due to a single agent (or few agents) at the highest level of the agent organization hierarchy. To avoid this problem, agents in future CPSs may follow, for example, a *distributed* interaction pattern where they self-organize to optimize their interaction or to accommodate to changes in the organizational view of the physical world. In this case, agent roles need to become *transferable*, i.e., delegable to various agents at run-time. For such a distributed setting, the presented PRESCRIPTIVECPS is not sufficient and needs to be further extended.

Addressing these challenges is future work. More future work is overviewed in the next section.

12.3 Future Work

There is plenty of future work to be done at all levels of a CPS system. This includes immediate work to be done in the scope of PRESCRIPTIVECPS as well as work to be done for planning-capable CPSs in general. We now review future work in these categories individually.

Immediate future work on PrescriptiveCPS

At the PRESCRIPTIVECPS component level, future work may consider:

- Different flexibility and prescription model storage approaches (e.g., using files, model-repositories, or no-SQL/array-based/main-memory databases) and their

comparison to the integrated relational DBMS approach, presented in this thesis. Also, processing flexibility and prescription models as *streams*, *scientific*, or *Hadoop-based* workflows is an interesting future work.

- Novel flexibility model aggregation and disaggregation techniques that automatically adjust the losses of a stimuli and responses to, for example, momentary agent system loads or new planning objectives.
- The implementation of the presented forecasting-capable DBMS and its experimental evaluation, for practically showing advantages, gained with respect to the traditional time series forecasting approach.
- New query optimization techniques that optimize queries involving what-if analysis, forecasting, and optimization problem solving. The design and the experimental evaluation of a DBMS with such query optimization capabilities is an interesting future work.
- A comprehensive evaluation of the RT-capable planning component integrating all the presented *storage* and *query processing* optimizations techniques.

At the PRESCRIPTIVECPS agent level, future work may consider:

- The implementation of an agent software, developed according to the proposed design, and its experimental evaluation in a simulated PRESCRIPTIVECPS environment, where *time*, *message exchange*, *physical sensing/actuation* are simulated according to user-defined simulation (incl., stressful) scenarios.

At the CP level, future work may consider:

- The prototype of the complete PRESCRIPTIVECPS, for studying the joint effect of many interacting agents and their joint capability to plan and control the behaviour of a concrete physical system (or processes).

Future work on CPSs in general

Planning under uncertainty, where stimuli and responses in flexibility models are defined as probability distributions, is an interesting future work, which considers the use of PRESCRIPTIVECPS in hardly *foreseeable* and *accessible* physical environments (see Figure 12.2).

PRESCRIPTIVECPS does not consider many other aspects related of practical use of a CPSs with planning capabilities. Future work may consider different agent *hardware architectures*, *communication protocols*, and *standards* as well as the resilience to *failures*, *malicious behaviour*, and *attacks* (e.g., man-in-the-middle, DDoS).

Privacy and *security* of data as well as the integration of various PRESCRIPTIVECPS instances to other existing systems (e.g., *social-networks*) are other directions of future work.

Dansk Resumé (Summary in Danish)

Mere og mere af vores fysiske verden bliver overvåget og kontrolleret af såkaldte cyber-fysiske systemer (CPSer). Disse er sammensætninger af netværksbaserede autonome IT (cyber) og fysiske (physical) agenter, såsom sensorer, aktuatorer, beregningsenheder, og mennesker. I dag er CPSer stadig forholdsvis små og meget begrænsede i forhold til de CPSer vi kan forvente i fremtiden. Fremtidige CPSer forventes at være langt mere komplekse, storstilede, udbredte, og missionskritiske, og vil kunne findes i en række områder såsom transport, medicin, produktion og energi, hvor de vil give mange fordele, såsom øget effektivitet, bæredygtighed, pålidelighed og sikkerhed. For at frigøre CPSernes fulde potentiale, skal de bl.a. udstyres med støtte til automatiseret planlægning og kontrol, hvor beregningsagenter i samspil og løbende planlægger og styrer deres handlinger på en intelligent og velkoordineret måde for at sikre og optimere en fysisk proces, såsom elforsyningen i elnettet.

I nuværende CPSer er styringen typisk automatiseret, mens planlægningen udelukkende er foretaget af mennesker. Det er umuligt for mennesker at planlægge hver handling i et fremtidigt CPS på grund af kompleksiteten, skalaen, og omskifteligheden af en fysisk proces. På grund af disse egenskaber, skal kontrol og planlægning være kontinuerlig og automatiseret i fremtidens CPSer. Mennesker kan kun analysere og justere systemets drift ved hjælp af det sæt af værktøjer, der understøtter præsriptive analyser (prescriptive analytics), der giver dem mulighed for (1) at lave forudsigelser, (2) at få forslagene fra de mest fremtrædende sæt handlinger (beslutninger), der skal tages, og (3) at analysere konsekvenserne, hvis sådanne handlinger blev udført.

Denne afhandling omhandler planlægning og kontrol i forbindelse med store multi-agent CPSer. Baseret på en smart-grid use case, præsenterer afhandlingen det såkaldte PrescriptiveCPS hvilket er (den konceptuelle model af) et multi-agent, multi-rolle, og multi-level CPS, der automatisk og kontinuerligt tager beslutninger i nær-realtid og leverer (menneskelige) brugere præsriptiveanalyseværktøjer til at analysere og håndtere det underliggende fysiske system (eller proces). I erkendelse af kompleksiteten af CPSer,

giver denne afhandling bidrag til følgende tre niveauer: (1) niveauet for et (fuldt) PrescriptiveCPS, (2) niveauet for en enkelt PrescriptiveCPS agent, og (3) niveauet for en komponent af et CPS agent software system.

På CPS-niveau, omfatter bidragene definitionen af PrescriptiveCPS, i henhold til hvilken det er det system med interagerende fysiske- og IT- (under-) systemer. Her består IT-systemet af hierarkisk organiserede forbundne agenter der sammen styrer instanser af såkaldte fleksibilitet (flexibility), beslutning (decision) og præskriptive (prescription) modeller, som henholdsvis er kortvarige, fokuserer på fremtiden, og repræsenterer en kapacitet, en (brugers) intention, og måder til at ændre adfærd (tilstand) af et fysisk system.

På agentniveau omfatter bidragene en tre-lags arkitektur af et agent software system, der integrerer antallet af komponenter, der er specielt konstrueret eller udbygges til at understøtte funktionaliteten af PrescriptiveCPS.

Komponentniveauet er hvor afhandlingen har sit hovedbidrag. Bidragene omfatter beskrivelse, design og eksperimentel evaluering af (1) et samlet multi- dimensionelt skema til at opbevare fleksibilitet og præskriptive modeller (og data), (2) teknikker til trinvis aggregering af fleksibilitet modelinstanser og disaggregering af præskriptive modelinstanser (3) et database management system (DBMS) med indbygget optimeringsproblemløsning (optimization problem solving) der gør det muligt at formulere optimeringsproblemer ved hjælp af SQL-lignende forespørgsler og at løse dem "inde i en database", (4) en realtids data management arkitektur til at behandle instanser af fleksibilitet og præskriptive modeller under (bløde eller hårde) tidsbegrænsninger, og (5) en grafisk brugergrænseflade (GUI) til visuelt at analysere fleksibilitet og præskriptive modelinstanser. Derudover diskuterer og eksemplificerer afhandlingen (men giver ingen evalueringer af) (1) domæne-specifikke og in-DBMS generiske prognosemetoder der gør det muligt at forudsige instanser af fleksibilitet modeller baseret på historiske data, og (2) kraftfulde måder at analysere tidligere-, nutids- og fremtidsbaserede såkaldte hypotetiske hvad-hvis scenarier og fleksibilitet og præskriptive modelinstanser gemt i en database. De fleste af bidragene på dette niveau er baseret på et smart-grid brugsscenarie.

Sammenfattende giver afhandlingen (1) modellen for et CPS med planlægningsmulighed, (2) design og eksperimentel evaluering af præskriptive analyse teknikker der gør det muligt effektivt at forudsige, aggregere, disaggregere, visualisere og analysere komplekse modeller af den fysiske verden, og (3) brugsscenariet fra energiområdet, der viser, hvordan de indførte begreber kan anvendes i den virkelige verden. Vi mener, at dette bidrag udgør et betydeligt skridt i retning af at udvikle CPSer til planlægningsbrug i fremtiden.

Zusammenfassung (Summary in German)

Mehr und mehr wird heute unsere physische Welt überwacht und durch sogenannte Cyber-Physical-Systems (CPS) geregelt. Dies sind Kombinationen von vernetzten autonomen cyber und physischen Agenten wie Sensoren, Aktoren, Rechelementen und Menschen. Heute sind CPS noch relativ klein und im Vergleich zu CPS der Zukunft sehr begrenzt. Zukünftige CPS werden voraussichtlich weit komplexer, größer, weit verbreiteter und unternehmenskritischer sein sowie in einer Vielzahl von Bereichen wie Transport, Medizin, Fertigung und Energie – in denen sie viele Vorteile wie erhöhte Effizienz, Nachhaltigkeit, Zuverlässigkeit und Sicherheit bringen – anzutreffen sein. Um ihr volles Potenzial entfalten zu können, müssen CPS unter anderem mit der Unterstützung automatisierter Planungs- und Steuerungsfunktionalität ausgestattet sein, so dass Agents ihre Aktionen gemeinsam und kontinuierlich auf intelligente und gut koordinierte Weise planen und kontrollieren können, um einen physischen Prozess wie den Stromfluss im Stromnetz sicherzustellen und zu optimieren.

Zwar sind in den heutigen CPS Steuerung und Kontrolle typischerweise automatisiert, aber die Planung wird weiterhin allein von Menschen durchgeführt. Leider ist diese Aufgabe nur schwer zu bewältigen, und es ist für den Menschen schlicht unmöglich, jede Aktion in einem zukünftigen CPS auf Basis der Komplexität, des Umfangs und der Volatilität eines physikalischen Prozesses zu planen. Aufgrund dieser Eigenschaften müssen Steuerung und Planung in CPS der Zukunft kontinuierlich und automatisiert ablaufen. Der Mensch soll sich dabei ganz auf die Analyse und Einflussnahme auf das System mit Hilfe einer Reihe von Werkzeugen konzentrieren können. Derartige Werkzeuge erlauben (1) Vorhersagen, (2) Vorschläge der wichtigsten auszuführenden Aktionen (Entscheidungen) und (3) die Analyse und potentiellen Auswirkungen der zu fällenden Entscheidungen.

Diese Arbeit beschäftigt sich mit der Planung und Kontrolle im Rahmen großer Multi-Agent-CPS. Basierend auf dem Smart-Grid als Anwendungsfall wird ein sogenanntes PrescriptiveCPS vorgestellt, welches einem Multi-Agent-, Multi-Role- und Multi-Level-CPS bzw. dessen konzeptionellem Modell entspricht. Diese PrescriptiveCPS treffen und realisieren automatisch und kontinuierlich Entscheidungen in naher Echtzeit

und stellen Benutzern (Menschen) Prescriptive-Analytics-Werkzeuge und Verwaltung der Leistung der zugrundeliegenden physischen Systeme bzw. Prozesse zur Verfügung. In Anbetracht der Komplexität von CPS leistet diese Arbeit Beiträge auf folgenden Ebenen: (1) Gesamtsystem eines PrescriptiveCPS, (2) PrescriptiveCPS-Agenten und (3) Komponenten eines CPS-Agent-Software-Systems.

Auf CPS-Ebene umfassen die Beiträge die Definition von PrescriptiveCPS als ein System von wechselwirkenden physischen und cyber (Sub-)Systemen. Das Cyber-System besteht hierbei aus hierarchisch organisierten verbundenen Agenten, die zusammen Instanzen sogenannter Flexibility-, Decision- und Prescription-Models verwalten, welche von kurzer Dauer sind, sich auf die Zukunft konzentrieren und Fähigkeiten, Absichten (des Benutzers) und Aktionen darstellen, die das Verhalten des physischen Systems verändern.

Auf Agenten-Ebene umfassen die Beiträge die Drei-Ebenen-Architektur eines Agentensoftwaresystems sowie die Integration von Komponenten, die insbesondere zur besseren Unterstützung der Funktionalität von PrescriptiveCPS entwickelt wurden.

Der Schwerpunkt dieser Arbeit bilden die Beiträge auf der Komponenten-Ebene, diese umfassen Beschreibung, Design und experimentelle Evaluation (1) eines einheitlichen multidimensionalen Schemas für die Speicherung von Flexibility- and Prescription-Models (und verwandten Daten), (2) der Techniken zur inkrementellen Aggregation von Instanzen eines Flexibilitätsmodells und Disaggregation von Prescription-Models, (3) eines Datenbankmanagementsystem (DBMS) mit integrierter Optimierungskomponente, die es erlaubt, Optimierungsprobleme mit Hilfe von SQL-ähnlichen Anfragen zu formulieren und sie „in einer Datenbank zu lösen“, (4) einer Echtzeit-Datenmanagementarchitektur zur Verarbeitung von Instanzen der Flexibility- and Prescription-Models unter (weichen oder harten) Zeitvorgaben und (5) einer grafische Benutzeroberfläche (GUI) zur Visualisierung und Analyse von Instanzen der Flexibility- and Prescription-Models. Darüber hinaus diskutiert und veranschaulicht diese Arbeit beispielhaft ohne detaillierte Evaluation (1) anwendungsspezifische und im DBMS integrierte Vorhersageverfahren, die die Vorhersage von Instanzen der Flexibility- and Prescription-Models auf Basis historischer Daten ermöglichen, und (2) leistungsfähige Möglichkeiten zur Analyse von Vergangenheit, Gegenwart und Zukunft auf Basis sogenannter hypothetischer „What-if“-Szenarien und der in der Datenbank hinterlegten Instanzen der Flexibility- and Prescription-Models. Die meisten der Beiträge auf dieser Ebene basieren auf dem Smart-Grid-Anwendungsfall.

Zusammenfassend befasst sich diese Arbeit mit (1) dem Modell eines CPS mit Planungsfunktionen, (2) dem Design und der experimentellen Evaluierung von Prescriptive-Analytics-Techniken, die eine effektive Vorhersage, Aggregation, Disaggregation, Visualisierung und Analyse komplexer Modelle der physischen Welt ermöglichen und (3) dem Anwendungsfall der Energiedomäne, der zeigt, wie die vorgestellten Konzepte in der Praxis Anwendung finden. Wir glauben, dass diese Beiträge einen wesentlichen Schritt in der zukünftigen Entwicklung planender CPS darstellen.

References

- [1] M. Boehm, L. Dannecker, A. Doms, E. Dovgan, B. Filipič, U. Fischer, W. Lehner, T. B. Pedersen, Y. Pitarch, L. Šikšnys, and T. Tušar, “Data management in the MIRABEL smart grid system,” in *Proc. of EDBT/ICDT Workshops*, 2012, pp. 95–102.
- [2] L. Šikšnys, C. Thomsen, and T. B. Pedersen, “MIRABEL DW: managing complex energy data in a smart grid,” in *Proc. of DaWaK*, 2012, pp. 443–457.
- [3] L. Šikšnys, M. E. Khalefa, and T. B. Pedersen, “Aggregating and disaggregating flexibility objects,” in *Proc. of SSDBM*, 2012, pp. 379–396.
- [4] T. Tušar, L. Šikšnys, T. B. Pedersen, E. Dovgan, and B. Filipič, “Using aggregation to improve the scheduling of flexible energy offers,” in *Proc. of BIOMA*, 2012, pp. 347–358.
- [5] U. Fischer, L. Dannecker, L. Šikšnys, F. Rosenthal, M. Böhm, and W. Lehner, “Towards Integrated Data Analytics: Time Series Forecasting in DBMS,” *Datenbank-Spektrum*, vol. 13, no. 1, pp. 45–53, 2013.
- [6] D. Kaulakienė, L. Šikšnys, and Y. Pitarch, “Towards the automated extraction of flexibilities from electricity time series,” in *Proc. of EDBT/ICDT Workshops*, 2013, pp. 267–272.
- [7] L. Šikšnys and T. B. Pedersen, “SolveDB: Solving Optimization Problems In the Database,” in *submission*, n/a.
- [8] U. Fischer, D. Kaulakienė, M. E. Khalefa, W. Lehner, T. B. Pedersen, L. Šikšnys, and C. Thomsen, “Real-time Business Intelligence in the MIRABEL Smart Grid System,” in *Proc. of BIRTE*, 2012, pp. 1–22.
- [9] L. Šikšnys and D. Kaulakienė, “Visualizing complex energy planning objects with inherent flexibilities,” in *Proc. of EDBT/ICDT Workshops*, 2013, pp. 249–255.

- [10] R. Rajkumar, I. Lee, L. Sha, and J. A. Stankovic, "Cyber-physical systems: the next computing revolution." in *Proc. of DAC*, 2010, pp. 731–736.
- [11] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., 2004.
- [12] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.
- [13] P. J. Haas, P. P. Maglio, P. G. Selinger, and W. C. Tan, "Data is Dead... Without What-If Models," *PVLDB*, vol. 4, no. 12, pp. 1486–1489, 2011.
- [14] L. V. S. Lakshmanan, A. Russakovsky, and V. Sashikanth, "What-if OLAP Queries with Changing Dimensions," in *Proc. of ICDE*, 2008, pp. 1334–1336.
- [15] A. Basu, "Five pillars of prescriptive analytics success," *Analytics*, pp. 8–12, 2013.
- [16] E. Guizzo. (2011) How Google's Self-Driving Car Works. Online. [Online]. Available: <http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works>
- [17] C. Killian, *Modern Control Technology*. Thomson Delmar Learning, 2005.
- [18] M. E. desJardins, E. H. Durfee, C. L. Ortiz, and M. J. Wolverton, "A Survey of Research in Distributed, Continual Planning," *AI Magazine*, vol. 20, no. 4, 1999.
- [19] G. Weiss, Ed., *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, 1999.
- [20] S. J. Russell, P. Norvig, J. F. Candy, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., 1996.
- [21] M. M. Veloso, "Towards Mixed-Initiative Rationale-Supported Planning," in *Proc. of Advanced Planning Technology*, 1996, pp. 277–282.
- [22] G. Ferguson and J. F. Allen, "Mixed-Initiative Systems for Collaborative Problem Solving," *AI Magazine*, vol. 28, no. 2, pp. 23–32, 2007.
- [23] A. Bharadwaj, J. Choobineh, A. Lo, and B. Shetty, "Model management systems: A survey," *Annals of Operations Research*, vol. 38, no. 1, pp. 17–67, 1992.
- [24] H. Geffner and B. Bonet, "A Concise Introduction to Models and Methods for Automated Planning," *Synthesis AIM*, vol. 8, no. 1, pp. 1–141, 2013.
- [25] *The MIRABEL Project*, 2012, <http://www.mirabel-project.eu>.

- [26] E. W. E. Association, “Creating the Internal Energy Market in Europe,” EWEA, Tech. Rep., 2012.
- [27] ETSO, “The Harmonized Electricity Market Role Model,” https://www.entsoe.eu/fileadmin/user_upload/edi/library/role/role-model-v2011-01.pdf, 2009.
- [28] W. A. Muhanna and R. A. Pick, “Meta-modeling concepts and tools for model management: a systems approach,” *Manage. Sci.*, vol. 40, no. 9, pp. 1093–1123, 1994.
- [29] P.-A. Muller, F. Fondement, B. Baudry, and B. Combemale, “Modeling modeling modeling,” *Software & Systems Modeling*, vol. 11, no. 3, pp. 347–359, 2012.
- [30] J.-M. Favre and T. NGuyen, “Towards a Megamodel to Model Software Evolution through Transformations,” in *Proc. of SETRA Workshop*, 2004, pp. 59–74.
- [31] J. F. Allen and J. A. Koomen, “Planning using a temporal world model,” in *Proc. of IJCAI*, 1983, pp. 741–747.
- [32] P. Stone and M. Veloso, “Multiagent Systems: A Survey from a Machine Learning Perspective,” *Auton. Robots*, vol. 8, no. 3, pp. 345–383, 2000.
- [33] E. A. Brewer, “Towards Robust Distributed Systems,” in *Proc. of PODC*, 2000, p. 7.
- [34] M. S. Wisz, R. J. Hijmans, J. Li, A. T. Peterson, C. H. Graham, A. Guisan, and N. P. S. D. W. Group, “Effects of sample size on the performance of species distribution models,” *Diversity and Distributions*, vol. 14, no. 5, pp. 763–773, 2008.
- [35] T. Tusar, E. Dovgan, and B. Filipic, “Evolutionary scheduling of flexible offers for balancing electricity supply and demand,” in *Proc. of IEEE CEC*, 2012, pp. 1–8.
- [36] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.
- [37] A. Balmin, T. Papadimitriou, and Y. Papakonstantinou, “Hypothetical Queries in an OLAP Environment,” in *Proc. of VLDB*, 2000, pp. 220–231.
- [38] D. Lomet, R. Barga, and R. Wang, “Transaction Time Support Inside a Database Engine,” in *Proc. of ICDE*, 2006, p. 35.
- [39] V. Jack, R. Diederik, R. Frens-Jan, and K. Mente, “D7.5 MIRABEL-ONE: Initial draft of the MIRABEL Standard,” <http://www.db.inf.tu-dresden.de/miracle/publications/D7.5.pdf>, 2011.

- [40] C. S. Jensen, T. B. Pedersen, and C. Thomsen, “Multidimensional Databases and Data Warehousing,” *Synthesis Lectures on Data Management*, vol. 2, no. 1, pp. 1–111, 2010.
- [41] www.meregio.de/en/, as of 2012-03-31.
- [42] postgresql.org, as of 2012-03-31.
- [43] “IEC 61970: Energy management system application program interface (EMS-API) - Part 301: Common information model (CIM) base, Third Edition,” 2009.
- [44] “EBIX: Introduction to Business Requirements and Information Models,” 2009.
- [45] J. Zubcoff, J. Pardillo, and J. Trujillo, “A UML Profile for the Conceptual Modelling of Data-mining with Time-series in Data Warehouses,” *Inf. Softw. Technol.*, vol. 51, no. 6, pp. 977–992, 2009.
- [46] M. Böhlen, J. Gamper, and C. S. Jensen, “Multi-dimensional aggregation for temporal data,” in *Proc. of EDBT*, 2006, pp. 257–275.
- [47] E. Malinowski, *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications*. Springer-Verlag, 2009.
- [48] A. Bauer, W. Hümmer, and W. Lehner, “An Alternative Relational OLAP Modeling Approach,” in *Proc. of DaWaK*, 2000, pp. 189–198.
- [49] I.-Y. Song, I. yeol Song, C. Medsker, E. Ewen, and W. Rowen, “An Analysis of Many-to-Many Relationships Between Fact and Dimension Tables in Dimensional Modeling,” in *Proc. of DMDW*, 2001, pp. 6–1.
- [50] M. Yue, “A simple proof of the inequality $FFD(L) \leq \frac{11}{9}OPT(L) + 1, \forall L$ for the FFD bin-packing algorithm,” *Acta Mathematicae Applicatae Sinica*, vol. 7, no. 4, pp. 321–331, 1991.
- [51] Y. N. Silva, A. M. Aly, W. G. Aref, and P.-A. Larson, “SimDB: A similarity-aware database system,” in *Proc. of SIGMOD*, 2010, pp. 1243–1246.
- [52] T. Tusar, E. Dovgan, and B. Filipic, “Evolutionary scheduling of flexible offers for balancing electricity supply and demand,” in *Proc. of IEEE CEC*, 2012, pp. 1–8.
- [53] T. Zhang, R. Ramakrishnan, and M. Livny, “BIRCH: an efficient data clustering method for very large databases,” in *Proc. of SIGMOD*, 1996, pp. 103–114.
- [54] J. B. Macqueen, “Some methods of classification and analysis of multivariate observations,” in *Proc. of 5th Berkeley Symposium on Math. Stat. and Prob.*, 1967, pp. 281–297.

- [55] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30–34, Jan. 1973.
- [56] Z. Zhang, Y. Yang, A. K. H. Tung, and D. Papadias, "Continuous k-means Monitoring over Moving Objects," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 9, pp. 1205–1216, 2008.
- [57] C. S. Jensen, D. Lin, and B. C. Ooi, "Continuous Clustering of Moving Objects," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 9, pp. 1161–1174, 2007.
- [58] N. H. Park and W. S. Lee, "Statistical grid-based clustering over data streams," *SIGMOD Rec.*, vol. 33, no. 1, pp. 32–37, 2004.
- [59] G. Hou, R. Yao, J. Ren, and C. Hu, "A clustering algorithm based on matrix over high dimensional data stream," in *Proc. of WISM*, 2010, pp. 86–94.
- [60] G. Lei, X. Yu, X. Yang, and S. Chen, "An incremental clustering algorithm based on grid," in *Proc. of FSKD*, 2011, pp. 1099–1103.
- [61] Y. N. Silva, W. G. Aref, and M. H. Ali, "Similarity Group-By," in *Proc. of ICDE*, 2009, pp. 904–915.
- [62] E. Malinowski and E. Zimnyi, *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications*, 1st ed. Springer, 2008.
- [63] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson, "A foundation for capturing and querying complex multidimensional data," *Information Systems*, vol. 26, no. 5, pp. 383–423, Jul 2001.
- [64] J. Cabot, J.-N. Mazón, J. Pardillo, and J. Trujillo, "Specifying aggregation functions in multidimensional models with OCL," in *Proc. of ER*, 2010, pp. 419–432.
- [65] D. Zhang, "Aggregation computation over complex objects," Ph.D. dissertation, University of California, Riverside, 2002.
- [66] M. H. Böhlen, J. Gamper, and C. S. Jensen, "How Would You Like to Aggregate Your Temporal Data?" in *Proc. of TIME*, 2006, pp. 121–136.
- [67] J. Yang and J. Widom, "Incremental computation and maintenance of temporal aggregates," *VLDB*, vol. 12, no. 3, pp. 262–283, Oct 2003.
- [68] A. Arasu and J. Widom, "Resource Sharing in Continuous Sliding-Window Aggregates," in *Proc. of VLDB*, 2004, pp. 336–347.
- [69] C. Jin and J. G. Carbonell, "Incremental Aggregation on Multiple Continuous Queries," in *Proc. of ISMIS*, 2006, pp. 167–177.

- [70] C.-Y. Chow, M. F. Mokbel, and T. He, “Aggregate Location Monitoring for Wireless Sensor Networks: A Histogram-Based Approach,” in *Proc. of MDM*, 2009, pp. 82–91.
- [71] M. H. Böhlen, J. Gamper, and C. S. Jensen, “Multi-dimensional Aggregation for Temporal Data,” in *Proc. of EDBT*, 2006, pp. 257–275.
- [72] J. Gordevičius, J. Gamper, and M. Böhlen, “Parsimonious temporal aggregation,” in *Proc. of EDBT*, 2009, pp. 1006–1017.
- [73] D. Gao, J. A. G. Gendrano, B. Moon, R. T. Snodgrass, M. Park, B. C. Huang, and J. M. Rodrigue, “Main Memory-Based Algorithms for Efficient Parallel Aggregation for Temporal Databases,” *DAPD*, vol. 16, no. 2, pp. 123–163, 2004.
- [74] B. Moon, I. Fernando Vega Lopez, and V. Immanuel, “Efficient Algorithms for Large-Scale Temporal Aggregation,” *TKDE*, vol. 15, no. 3, pp. 744–759, 2003.
- [75] S. D., Y. Sismanis, K. S. Beyer, R. Gemulla, P. J. Haas, and J. McPherson, “Ricardo: Integrating R and Hadoop,” in *Prof. of SIGMOD*, 2010, pp. 987–998.
- [76] R. Ramanathan, R. Engle, C. W. J. Granger, F. Vahid-Araghi, and C. Brace, “Short-run Forecasts of Electricity Loads and Peaks,” *International Journal of Forecasting*, vol. 13, no. 2, pp. 161–174, 1997.
- [77] Oracle, *Oracle OLAP DML Reference: FORECAST - DML Statement*, 2011.
- [78] *PredictTimeSeries - Microsoft SQL Server 2008 Books Online*, 2011. [Online]. Available: <http://msdn.microsoft.com/en-us/library/ms132167.aspx>
- [79] S. Duan and S. Babu, “Processing Forecasting Queries,” in *Proc. of VLDB*, 2007, pp. 711–722.
- [80] F. Parisi, A. Sliva, and V. S. Subrahmanian, “Embedding Forecast Operators in Databases,” in *Proc. of SUM*, 2011, pp. 373–386.
- [81] T. Ge and S. Zdonik, “A Skip-list Approach for Efficiently Processing Forecasting Queries,” in *Proc. of VLDB*, 2008, pp. 984–995.
- [82] D. Agarwal, D. Chen, L. Lin, J. Shanmugasundaram, and E. Vee, “Forecasting High-dimensional Data,” in *Proc. of SIGMOD*, 2010, pp. 1003–1012.
- [83] U. Fischer, F. Rosenthal, M. Boehm, and W. Lehner, “Indexing Forecast Models for Matching and Maintenance,” in *Proc. of IDEAS*, 2010, pp. 26–31.
- [84] U. Fischer, M. Boehm, and W. Lehner, “Offline Design tuning for Hierarchies of Forecast Models,” in *Proc. of BTW*, 2011, pp. 167–186.

- [85] L. Dannecker, M. Boehm, W. Lehner, and G. Hackenbroich, “Forecasting Evolving Time Series of Energy Demand and Supply,” in *Proc. of ADBIS*, 2011, pp. 302–315.
- [86] F. Rosenthal and W. Lehner, “Efficient In-Database Maintenance of ARIMA Models,” in *Proc. of SSDBM*, 2011, pp. 537–545.
- [87] L. Dannecker, M. Boehm, W. Lehner, and G. Hackenbroich, “Partitioning and Multi-Core Parallelization of Multi-Equation Forecast Models,” in *Proc. of SSDBM*, 2012, pp. 106–123.
- [88] L. Dannecker, R. Schulze, M. Boehm, W. Lehner, and G. Hackenbroich, “Context-Aware Parameter Estimation for Forecast Models in the Energy Domain,” in *Proc. of SSDBM*, 2011, pp. 491–508.
- [89] W. Lehner, *Datenbanktechnologie für Data-Warehouse-Systeme. Konzepte und Methoden*. dpunkt, 2003.
- [90] N. Roussopoulos, “The Logical Access Path Schema of a Database,” *IEEE Trans. Softw. Eng.*, vol. 8, no. 6, pp. 563–573, 1982.
- [91] I. Sánchez, “Adaptive combination of forecasts with application to wind energy,” *International Journal of Forecasting*, vol. 24, no. 4, pp. 679 – 693, 2008.
- [92] F. Rosenthal, P.-B. Volk, M. Hahmann, D. Habich, and W. Lehner, “Drift-Aware Ensemble Regression,” in *Proc. of MLDM*, 2009, pp. 221–235.
- [93] J. W. Taylor, “Triple Seasonal Methods for Short-term Electricity Demand Forecasting,” *EJOR*, vol. 204, no. 1, pp. 139–152, 2010.
- [94] U. Fischer, F. Rosenthal, and W. Lehner, “F2DB: The Flash-Forward Database System (Demo),” in *Proc. of ICDE*, 2012.
- [95] D. Dunn, W. Williams, and T. DeChaine, “Aggregate Versus Subaggregate Models in Local Area Forecasting,” *American Statistical Association*, vol. 71, pp. 68–71, 1976.
- [96] R. J. Hyndman, R. A. Ahmed, G. Athanasopoulos, and H. L. Shang, “Optimal combination forecasts for hierarchical time series,” *Computational Statistics and Data Analysis*, vol. 55, no. 9, pp. 2579 – 2589, 2011.
- [97] R. J. Hyndman and Y. Khandakar, “Automatic Time Series Forecasting: The forecast Package for R,” *Statistical Software*, vol. 27, 2008.

- [98] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose, “A State Space Framework For Automatic Forecasting Using Exponential Smoothing Methods,” Department of Econometrics and Business Statistics, Monash University, Tech. Rep., 2000.
- [99] J. Goldstein and P.-A. Larson, “Optimizing queries using materialized views: a practical, scalable solution,” in *Proc. of SIGMOD*, 2001, pp. 331–342.
- [100] A. M. Sharp, “Incremental algorithms: solving problems in a changing world,” Ph.D. dissertation, Cornell University, 2007.
- [101] R. Fourer and D. Orban, “DrAmpl: a meta solver for optimization problem analysis,” *Computational Management Science*, vol. 7, no. 4, pp. 437–463, 2010.
- [102] S. K. Smit and A. E. Eiben, “Comparing Parameter Tuning Methods for Evolutionary Algorithms,” in *Proc. of IEEE CEC*, 2009, pp. 399–406.
- [103] M. E. H. Pedersen and A. J. Chipperfield, “Simplifying particle swarm optimization,” *ASOC*, vol. 10, no. 2, pp. 618–628, 2010.
- [104] D. R. Dolk, “An introduction to model integration and integrated modeling environments,” *DSS*, vol. 10, no. 3, pp. 249–254, 1993.
- [105] Y.-C. Tsai, “Comparative analysis of model management and relational database management,” *Omega*, vol. 29, no. 2, pp. 157–170, 2001.
- [106] H. J. Greenberg and F. H. Murphy, “Views of mathematical programming models and their instances,” *DSS*, vol. 13, no. 1, pp. 3–34, 1995.
- [107] A. Kawaguchi and A. Nagel, *Linear Programming in Database*. In-Tech Press, 2008, ch. 19, pp. 339–354.
- [108] A. Atamtürk, E. L. Johnson, J. T. Linderoth, and M. W. P. Savelsbergh, “A Relational Modeling System for Linear and Integer Programming,” *OR*, vol. 48, no. 6, pp. 846–857, 2000.
- [109] G. Dantzig, *Linear Programming and Extensions*. Princeton University Press, 1963.
- [110] J. Choobihieh, “SQLMP: A Data Sublanguage for Representation and Formulation of Linear Mathematical Models,” *ORSA Journal of Computing*, vol. 3, no. 4, pp. 358–375, 1991.
- [111] A. Brodsky, N. Egge, and X. Wang, “Supporting Agile Organizations with a Decision Guidance Query Language,” *JMIS*, vol. 28, no. 4, pp. 39–68, 2012.

- [112] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar, “The MADlib analytics library: or MAD skills, the SQL,” *PVLDB*, vol. 5, no. 12, pp. 1700–1711, 2012.
- [113] C. Ordonez, “Statistical Model Computation with UDFs,” *TKDE*, vol. 22, no. 12, pp. 1752–1765, 2010.
- [114] X. Feng, A. Kumar, B. Recht, and C. Ré, “Towards a unified architecture for in-RDBMS analytics,” in *Proc. of SIGMOD*, 2012, pp. 325–336.
- [115] H. K. Bhargava, R. Krishnan, and R. Müller, “Electronic Commerce in Decision Technologies: A Business Cycle Analysis,” *Int. J. Electron. Commerce*, vol. 1, no. 4, pp. 109–127, 1997.
- [116] P. Baybutt, “Major hazards analysis: An improved method for process hazard analysis,” *Process Safety Progress*, vol. 22, no. 1, pp. 21–26, 2003.
- [117] S. Ghandeharizadeh, R. Hull, and D. Jacobs, “Heraclitus: Elevating Deltas to Be First-class Citizens in a Database Programming Language,” *Trans. Database Syst.*, vol. 21, no. 3, pp. 370–426, 1996.
- [118] T. Griffin and R. Hull, “A Framework for Implementing Hypothetical Queries,” in *Proc. of SIGMOD*, 1997, pp. 231–242.
- [119] D. Deutch, Z. G. Ives, T. Milo, and V. Tannen, “Caravan: Provisioning for What-If Analysis,” in *Proc. of CIDR*, 2013.
- [120] R. Rajamani, “Oracle total recall / flashback data archive,” in *Oracle, Tech*, 2007.
- [121] K. Kulkarni and J.-E. Michels, “Temporal Features in SQL:2011,” *SIGMOD Rec.*, vol. 41, no. 3, pp. 34–43, 2012.
- [122] C. Thomsen, T. B. Pedersen, and W. Lehner, “RiTE: Providing On-Demand Data for Right-Time Data Warehousing,” in *Proc. of ICDE*, 2008, pp. 456–465.
- [123] M. E. Khalefa, U. Fischer, T. B. Pedersen, and W. Lehner, “Model-based Integration of Past & Future in TimeTravel,” *VLDB*, vol. 5, no. 12, pp. 1974–1977, 2012.
- [124] *Wind Integration Datasets*, NREL, 2012, <http://www.nrel.gov/wind/integrationdatasets/>.
- [125] E. Wu, Y. Diao, and S. Rizvi, “High-performance complex event processing over streams,” in *Proc. of SIGMOD*, 2006, pp. 407–418.

- [126] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, J. h. Hwang, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik, “Aurora: A Data Stream Management System,” in *Proc. of SIGMOD*, 2003, p. 666.
- [127] A. Kemper and T. Neumann, “HyPer: A Hybrid OLTP & OLAP Main Memory Database System Based on Virtual Memory Snapshots,” in *Proc. of ICDE*, 2011, pp. 195–206.
- [128] M. Thiele and W. Lehner, “Evaluation of Load Scheduling Strategies for Real-Time Data Warehouse Environments,” in *Proc. of BIRTE*, 2009, pp. 84–99.
- [129] S. Card, J. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., 1999.
- [130] H. Ltifi, M. B. Ayed, A. M. Alimi, and S. Lepreux, “Survey of information visualization techniques for exploitation in KDD,” in *Proc. of AICCSA*, 2009, pp. 218–225.
- [131] M. C. F. de Oliveira and H. Levkowitz, “From Visual Data Exploration to Visual Data Mining: A Survey,” *IEEE Trans. Vis. Comput. Graph.*, vol. 9, no. 3, pp. 378–394, 2003.
- [132] C. Ware, *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., 2004.
- [133] S. Few, *Now You See It: Simple Visualization Techniques for Quantitative Analysis*. Analytics Press, 2009.
- [134] S. Mansmann and M. H. Scholl, “Visual OLAP: A new paradigm for exploring multidimensional aggregates,” in *Proc. of IADIS*, 2008, pp. 59–66.
- [135] A. Cuzzocrea and S. Mansmann, “OLAP Visualization: Models, Issues, and Techniques,” *Encyclopedia of Data Warehousing and Mining*, pp. 1439–1446, 2009.
- [136] S. Lafon, F. Bouali, C. Guinot, and G. Venturini, “On studying a 3D user interface for OLAP,” *Data Mining and Knowledge Discovery*, vol. 27, no. 1, pp. 4–21, 2013.
- [137] H. Siirtola, “Interactive Visualization of Multidimensional Data,” Ph.D. dissertation, University of Tampere, 2007.
- [138] W. W.-Y. Chan, “A Survey on Multivariate Data Visualization,” Dept. of Comp. Sc. and Eng. HKUST, Tech. Rep., 2006.
- [139] D. Keim and H.-P. Kriegel, “Visualization techniques for mining large databases: a comparison,” *TKDE*, vol. 8, no. 6, pp. 923–938, 1996.

- [140] S. Silva and T. Catarci, “Visualization of linear time-oriented data: a survey,” in *Proc. of WISE*, 2000, pp. 310–319.
- [141] W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski, “Visualizing time-oriented data-A systematic view,” *Comput. Graph.*, vol. 31, no. 3, pp. 401–409, 2007.
- [142] M. Weber, M. Alexa, and W. Müller, “Visualizing Time-Series on Spirals,” in *Proc. of INFOVIS*, 2001, pp. 7–14.
- [143] M. Sips, P. Köthür, A. Unger, H.-C. Hege, and D. Dransch, “A Visual Analytics Approach to Multiscale Exploration of Environmental Time Series,” *TVCG*, vol. 18, no. 12, pp. 2899–2907, 2012.
- [144] J. Lin, E. Keogh, and S. Lonardi, “Visualizing and discovering non-trivial patterns in large time series databases,” *Information Visualization*, vol. 4, no. 2, pp. 61–82, 2005.
- [145] T. A. Slocum, R. B. McMaster, F. C. Kessler, and H. H. Howard, *Thematic Cartography and Geovisualization*. Prentice Hall, 2008.
- [146] R. Peikert, H. Hauser, H. Carr, and R. Fuchs, *Topological Methods in Data Analysis and Visualization II: Theory, Algorithms, and Applications*. Springer, 2012.
- [147] T. Ropinski, S. Oeltze, and B. Preim, “Survey of glyph-based visualization techniques for spatial multivariate medical data,” *Computers & Graphics*, vol. 35, no. 2, pp. 392 – 401, 2011.
- [148] N. Andrienko, G. Andrienko, and P. Gatalsky, “Exploratory spatio-temporal visualization: an analytical review,” *JVLC*, vol. 14, no. 6, pp. 503–541, 2003.
- [149] T. Overbye and J. Weber, “Visualization of power system data,” in *Proc. of HICSS*, 2000, p. 4014.
- [150] C. Mikkelsen, J. Johansson, and M. Rissanen, “Interactive Information Visualization for Sensemaking in Power Grid Supervisory Systems,” in *Proc. of IV*, 2011, pp. 119–126.
- [151] Z. Zhou, W. Wu, X. Dong, B. Zhang, and H. Sun, “A preliminary investigation on smart grid Operation Cockpit,” in *Proc. of ISGT Asia*, 2012, pp. 1–4.

Statement of Authorship

I hereby declare that I have written this thesis individually and without the use of *documents* other than those referenced in this thesis and *inputs* others than those which I acknowledge in this thesis and explicitly mention in the presented co-author statements (available with the thesis). Up to now, the work is submitted only at TU Dresden and the IT4BI-DC partner university AAU. The work has not been presented in this or a similar form to another examination agency neither in Germany nor in any other country, and it has not yet been published either.

Laurynas Šikšnys
February 24, 2014