Modeling Medication Prescriptions and Adherence

by

Simon Diemert

Partial Fulfillment of the
Requirements for the Degree of

Bachelors of Software Engineering

in the Faculty of Engineering

Modeling Medication Prescriptions and Adherence

by

Simon Diemert

Supervisory Committee

Dr. Jens Weber, Supervisor
(Department of Computer Science)

Dr. Morgan Price, Supervisor
(Department of Computer Science)

ABSTRACT

In modern medicine medication prescriptions are one of the primary forms of intervention. Prescriptions are effectively a specification for how to consume a particular kind of substance. A prescription's complexity can range from a simple verbal orders to intricate sequences of actions that vary depending on a number of different conditions. The adoption of Health Information Technology (HIT) has given rise to electronic prescribing (e-prescribing). Much of the functionality available e-prescribing systems requires that prescriptions be entered in a discrete and well-structured format. Unfortunately, these systems typically only capture a subset of the concepts of prescribing, while the rest of the concepts are entered as free text. The result is poor data quality which affects the quality of the decision support provided by the HIT system.

Advances in consumer technology have also allowed for the ability to measure how patients take medications. Such technologies are aimed at addressing the concern of medication adherence, where a patient does not adhere to their prescribed medication regimes. Non-adherence can lead to serious medical problems, and has been identified by numerous authorities and academics as a factor effecting the health care outcomes of both individuals and populations.

This thesis attempts to address parts of both e-Prescribing data quality and medication adherence. A Domain Specific Language (DSL) for authoring prescriptions was created. The goal of the DSL is to provide a structured means of creating medication prescriptions that is similar to the natural language expression of prescriptions used by many clinicians. The textual input from the DSL is transformed into a graph model. Once in graph form, the prescription can be manipulated using a computational technique called graph transformation that reduces prescription to a core model. The core model describes the entire prescription in a series of Atomic Prescribed Medication Actions (APMAs). This core model can be thought of as a plan for the prescription. When the core model is combined with the patient's actual "execution" of the prescription (collected from medication adherence measuring device) the degree of medication adherence can be computed. Using these graph models and transformation systems provides formal definition for the semantics of the prescribing DSL used as input. Such formality provides a means of reasoning about prescriptions in a mathematical way that can then be used to provide feedback and support for clinical and patient users.

Contents

List of Figures

Chapter 1

Introduction

Electronic health information systems are in wide use in clinical medicine; their application promises advanced data management and processing methods that can inform those participating in the health care process. One of the key components of many of these systems is a method for prescribing medications to patients. Many electronic health information systems provide interfaces for managing prescription information. Prescribing medications is one of the ways that medical professionals actuate the care of their patients; since prescribing modules in information systems are a key component of this process, their interfaces and the supporting software infrastructure merit serious consideration.

Advances in consumer technologies have provided a plethora of devices by which individuals can measure and participate in their health care process. However, these technologies require connection to existing (and future) information systems before they can be used effectively in the care of patients. One facet of connecting these devices is providing a means of reconciling information from these devices and the information stored in health information systems.

1.1 Statement of Problem

Software in other sectors (transport, energy etc...) has been established as safety-critical, however the health care sector is only beginning to recognize the implications of software system errors/failures [51]. After the uptake of electronic systems for managing health information, new types of errors began to emerge [30, 44, 20]. There has been a movement to consider software in health care as a complex sociotechnical

systems [51, 32, 7]; since e-prescribing systems are one of the tools used to actuate the health care of patients, they are no exception to this [17]. As a result, e-prescribing systems need to designed from the start with safety in mind.

One of the emerging concerns regarding the use of electronic information systems is the difficulty to catalog and process natural language text input [14, 15]. It is acceptable for clinicians to wish to express their input to information systems in a natural language format that captures their intention and is easily understood by others. However, in the absence of advanced natural language processing methods, this poses a significant problem for the task of extracting this information into a structured form. It has been found that this trend is reflected in prescribing habits, specifically, that prescribers express important information about prescriptions in free text fields [33]. The rationale behind this is that user interfaces do not accommodate more advanced concepts required to express prescriptions, or that the interfaces are to cumbersome too use effectively [16, 23, 24, 26, 47].

In addition to problematic user interfaces, different software systems represent prescriptions with different models that are not necessarily tied to an underlying real world semantics. Without a concrete mapping to a set of core semantics, reasoning inside the information system about the prescription can be difficult to achieve in the general case. This has implications for the effectiveness of clinical decision support systems, which rely on structured information to provide suggestions to clinicians [15, 14].

One factor affecting the effects of medications on a patient's outcome is the degree to which the patient follows the prescribed medication regime, this has been termed "medication adherence" (or non-adherence) [31, 39]. A patient is adherent to a prescription if they take the prescribed medication as directed; non-adherence occurs when the patient deviates from the prescribed regime. A variety of methods to measure medication adherence have been proposed, however they typically act as sensors which require a human to interpret the adherence information prior to making decisions about prescriptions [27, 29, 46]. Creating methods of using the data collected from these devices to measure adherence is a complex problem.

In summary, the problem under consideration is:

- Current software for managing and creating prescriptions is not meeting the needs of clinicians, particularly in the domain of expressing the concepts in a prescription.

- Software to manage prescriptions is part of a complex sociotechnical system that actuates the health care of persons. As a result it must be designed with safety as a core property of the system.
- Models of prescriptions currently do not provide enough grounding in reality to allow for decision support systems to be used to their full potential.
- Methods for reconciling patterns of medication adherence are not currently automated.

1.2 Objective of Research

The objective of this work is to provide a possible solution to the issues stated above. Specifically it aims to acheive the following goals:

- Provide a means of writing electronic prescriptions for clinicians that is reliable, expressive, and well structured.
- Develop a model that is flexible enough to express the complex notions used in prescribing medications, yet structured enough to allow for computation.
- Provide a model that formally describes the semantics of the domain specific language.
- Develop a formal definition of medication adherence that is grounded in the mathematical formalism of graph theory and transformation.
- Provide a means of reconciling the patient's execution of a prescription with the plan described by the prescription, thus providing a means of measuring medication adherence.

This work is meant to be a proof of concept for its models and methods. Effort was spent ensuring the subset of prescribing concepts that were considered were developed completely. As a result, this work has been limited to the context of primary care prescribing, rather than all possible contexts of prescriptions.

1.3 Summary of Approach

To address the issues described above, a core or most basic data model of a prescription was developed that can capture the complexity of prescribing in primary care. This model drove the development of a tool that can be used to author and then process prescriptions. The resulting software system could be made to plugin into an electronic prescribing module of a health information system.

To find a middle ground between expressibility of prescribing concepts and ease of use for inputting prescription information into the system, a Domain Specific Language (DSL) for writing prescriptions was created using ANTLR, a tool for creating DSLs. The parse tree of the DSL (after mild pre-processing) is used as input to a graph transformation (GT). The GT system distills the input to down to the core prescribing graph model.

Another graph model can be used to represent the administration of the medication over time. This graph model can be generated from any number of sensors in the patient's environment. This administration graph may or may not be adherent to the prescribed regime.

The execution trace and core model are reconciled by a third graph transformation system that determines to what the degree the trace is adherent to the original prescription. Figure 1.1 shows a high level block diagram of the system.

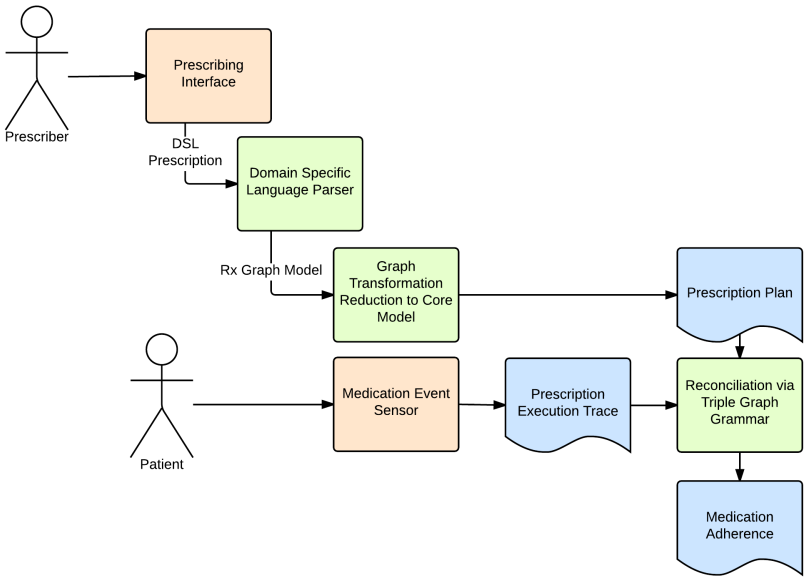


Figure 1.1: High level system model.

1.4 Agenda

The remainder of this work is structured as follows:

- **Chapter 2** provides relevant background material on the topics of prescribing, medication adherence, graph transformation, and domain specific languages.
- **Chapter 3** describes details of the domain specific language for prescribing.
- **Chapter 4** describes two graph models for representing prescriptions and a method for producing one from the other.
- **Chapter 5** describes how the core and administration graph models are reconciled using graph transformation.
- **Chapter 6** provides a discussion of the concepts presented in the proceeding chapters.
- **Chapter 7** examines related work on e-prescribing, and graph transformation.
- **Chapter 8** provides conclusions and future directions.
- **Appendix A** details the BNF of the prescribing DSL.
- **Appendix B** contains the rules used within the graph transformation systems.

Chapter 2

Background

This chapter aims to give relevant background for readers who may not be familiar with particular topics as they relate to the rest of this work. This chapter is not meant to give a survey of other work in the respective fields. Chapter 7 provides an informal review of current literature and work in the various fields.

2.1 Prescribing

Oxford's *Concise Medical Dictionary* defines a "prescription" as "a written direction from a registered medical practitioner to a pharmacist for preparing and dispensing a drug" [1]. Medication prescriptions are one of the main methods health care professionals use to actuate the health of their patients. In primary care, prescribing medications can be broken into three categories:

- Regular prescription medications
- Controlled substances
- Over the counter medications

Figure 2.1 describes a prescription process involving a patient, physician, and pharmacist. It can be described as:

- the patient requires care for a medical concern.

[1] Oxford University Press, http://www.oxfordreference.com/10.1093/acref/9780199557141.001.0001/acref-9780199557141-e-8167

- the patient seeks the care of a physician and receives a written prescription for medication to address their medical concern.
- the patient presents the prescription from their physician to the pharmacist at the pharmacy and collects their prescription.

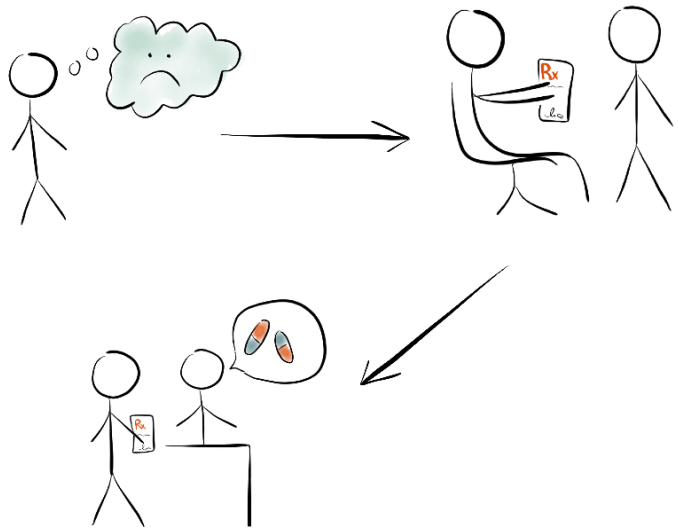


Figure 2.1: Description of the medication prescribing process.

The process outlined above should be familiar to most people who have required some form of prescription medication. *Prescription* medications require a document (called a "prescription", often shortened to "Rx") from a registered medical practitioner before they can be dispensed by a pharmacist. For example, Amoxicillin is an antibiotic medication that requires a prescription before it can be obtained by the patient. This process enforces a moderate amount of control over how prescription medications are dispensed and administered.

Some medications require special prescriptions, these often these are more tightly controlled substances. For example, Methadone is a controlled substance used to assist with the treatment of opioid dependency. Practitioners require special licensing to prescribe such substances. The dispensing pattern (patient, physician, pharmacist) for controlled substances is often similar to that of regular prescription medications,

however it may vary slightly depending on the substance, the practitioner, or the patient.

Prescriptions for "over the counter" medications, are not formally written prescriptions in the way that the previous two types are. Such prescriptions are often informal instructions from a practitioner for a patient to consume a medicine that does not require a pharmacist to dispense. Aspirin is an example of a substance that does not require any formal prescription from a registered practitioner, and can be purchased at many pharmacies, or supermarkets. However, this lack of control does not mean that practitioners do not need to be aware of their patient's use of such substances. Often these medications can conflict with other medications a patient may be prescribed; the practitioner needs to be aware of this to prevent harm to the patient. We consider these types of medications as prescriptions here as they are often listed as such in electronic medical records.

For completeness, prescriptions can also be written for other kinds of medical treatments or devices, however these are out of the scope of this work.

2.1.1 Prescription Structure

The core components that make up a prescription are as follows:

1. A **substance** to be used in the prescription (sometimes many substances are combined). Often referred to using a generic name ("Azithromycin"), or a brand name ("Zithromax").

2. The **route** of delivery describing how the substance is to be delivered ("oral", "topical", "subcutaneously", etc.).

3. A **form** or formulation of the substance, often related to the route ("tablet", "liquid", etc.).

4. The **dose** or amount of substance, often expressed as a number followed by a unit describing mass or volume ("81 mg" or "10 mL").

5. A description of the **timing**, typically consisting of a **frequency** and a **duration**, which describes how the doses of substance should be consumed over time ("once daily for 10 days").

6. **Start** and **stop** dates to indicate when to consume the substance in question and when to stop.

7. A **prescription date** indicating the day the prescription was created.

8. A **dispense date** to indicate the day the prescription can be collected at a pharmacy.

9. The **quantity** of substance to be dispensed at one time, usually a product of the dose and timing elements.

10. A number of **repeats** or **refills** that specifies how many times the prescription can be re-collected at the pharmacy.

11. A set of **instructions** describing constraints that are placed on how the substance should be consumed ("do not take with alcohol", "take with meals").

The distinction between a *medication* and a *prescription* is subtle but important when creating a data models of these concepts. A *medication* describes the substance, formulation, route, and dose. A *prescription* describes the instructions for using the medication and typically includes all other fields listed above. This distinction becomes apparent when describing the medical history of a patient.For example a patient may have had several prescriptions issued for the same medication. Figure 2.2 summarizes the relationship between all of the core prescription components with an entity relationship diagram.

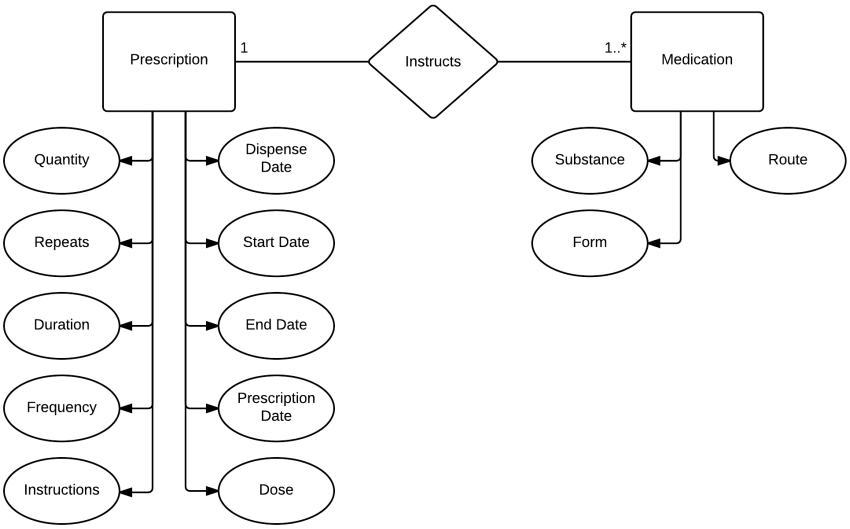


Figure 2.2: Entity relationship diagram of a prescription.

There are many ways in which the items (and others) can be combined to create a valid prescription, especially when the form of communication is natural language. For example, consider the following two prescriptions that are equivalent:

> *"Take azithromycin two 250 mg tablets on the first day followed by one 250 mg tablet for 4 more days."*

> *"Take 500 mg of azithromycin once, then take 250 mg once daily for four days."*

To complicate matters, the antibiotic Azithromycin can be purchased under the name brand "Zithromax", which contains administration instructions on the package. This antibiotic can then be prescribed as:

> *"Zithromax as directed."*

Clearly, these prescriptions do not explicitly state all of the components previously listed, however many of them are implied. For example, the prescriptions are for a **quantity** of 6 tablets total and the start date is the current day. Furthermore, medical practitioners have developed a short hand notation for many of these concepts, for example "fPO" indicates the **route** as "oral".

2.1.2 e-Prescribing

e-Prescribing is the practice of using an electronic medium to create and transfer prescriptions. e-Prescribing provides many benefits, some of which include better communication between health care professionals, advanced decision support at the point of care, and detailed patient histories. Theese and more are described in detail in [18].

The most common method of creating e-Prescriptions in primary care is to use a form based input method which contains fields for many of the items enumerated above. Figure 2.3 shows an example of the prescribing user interface used in OSCAR EMR [2].

As briefly mentioned in Chapter 1, interfaces like the one in Figure 2.3 restrict the expression of practitioners when they are creating prescriptions. The result is that important information about the prescription, usually in the form of "special

[2]Available at `http://oscar-emr.com/`

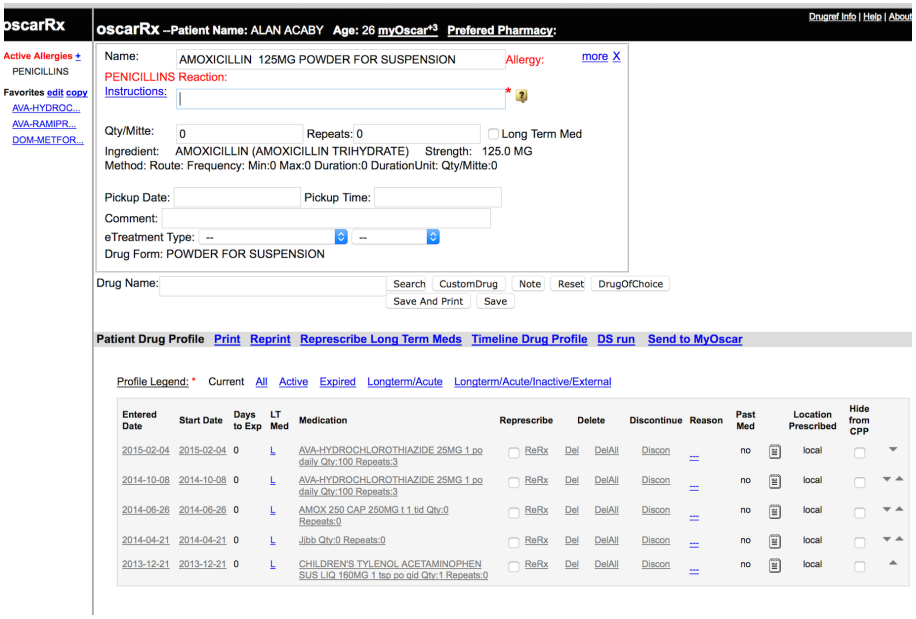


Figure 2.3: Sample e-prescribing user interface from OSCAR EMR 12.1

instructions", are entered as natural language in the "comment" field. This leads to a significant amount of clinically relevant data that is being entered to be ignored by the information system, which will reduce the amount of information that decision support tools will have to make suggestions to improve care. Thus finding methods to reduce the amount of "free text" or unstructured data are required to support other components of the electronic health record system.

2.2 Medication Adherence

Adherence to prescribed medication regimes is a key factor in improving patient outcomes in the health care systems [31, 49, 19]. Medication adherence is defined as the degree to which individuals follow instructions from their practitioners regarding their medications. The World Health Organization estimated that the international average for medication adherence is 50% and noted that improving medication adherence was potentially as important as developing new types of therapies [31].

Medication adherence is a non-trivial problem involving a myriad of factors that no one application or technique will be able solve [19]. A framework work for improving adherence must include *informed patients*, *patient motivation*, and *learning good adherence behaviours* [49]; methods aimed at improving medication adherence

should address each of these factors. A number of technical solutions to non-adherent behaviours have been proposed which range from pure software applications to large counter-top devices [29, 27, 48].

Non-adherent behaviour also puts a significant burden on the already stretched resources of the health care system. Non-adherent behaviour can lead to relapse or exacerbation of existing medical conditions or to the incidence of a new condition that requires additional resources to control. For example, consider a patient being treated for hypertension (high blood pressure). If that patient chooses not to take their prescribed anti-hypertensive medications, they increase their risk of a cardiovascular event, which will likely lead to hospitalization. Determining the exact cost of non-adherence behaviour is difficult due to the number of factors at play in any patient's health care; however, estimates have been made as high as $300 billion per year in the United States [49]. To compound the problem, self-reporting methods for measuring the degree of medication adherence have been shown to be inaccurate [50]; new methods of measuring medication adherence are required. To determine the degree of adherence, these methods must be able to compare the patient's execution of a prescription what the prescription actually says. This work provides a method of reconciling these two sets of prescription data which will support the measurement of adherence.

2.3 Graph Transformation

Graph transformation systems provide a means of defining manipulations to a graph. These systems have applications in a range of disciplines within the fields of Engineering, Computer Science, and Mathematics. The core concept in graph transformation is the concept of a graph, which provide a means of representing models of real-world scenarios. Transformation systems then specify a set of rules that manipulate the graph (often referred to as the "host graph"), the result of which is another graph. A variety of formalisms and methods for describing and executing graph transformations have been developed, including the GROOVE tool used for this work which is discussed in section 2.3.1. The remainder of this section describes some of the key concepts of graph transformation that are required for the reader to understand the remainder of this work. A more detailed and rigorous definition of graph transformation systems can be found in [10, 41].

Graphs

As mentioned previously, graphs can be used to describe models of real-world problems and phenomena. Graphs are a set of vertices (nodes) and edges that connect them. Formally, a graph can be defined as:

$$G = (V, E) \tag{2.1}$$

Where V is a set of vertices or nodes that make up the graph, and E is the set of edges that link the vertices. E is a set of ordered pairs of vertices, formally E is defined as:

$$E \subseteq V \times V \tag{2.2}$$

Consider the graph in Figure 2.5a; the vertex set of this graph is $V = \{A, B, C, D\}$ then the edge set is $E = \{(A, B), (A, C), (A, D), (B, D), (C, B), (D, B), (D, C)\}$.

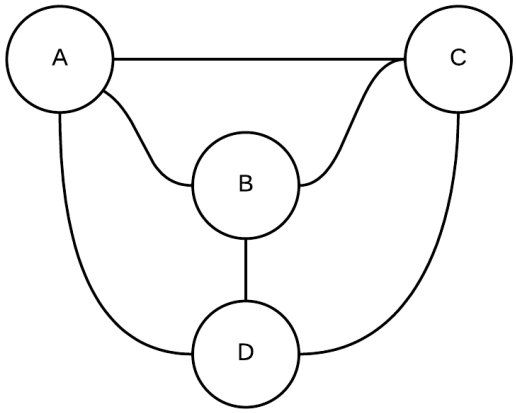
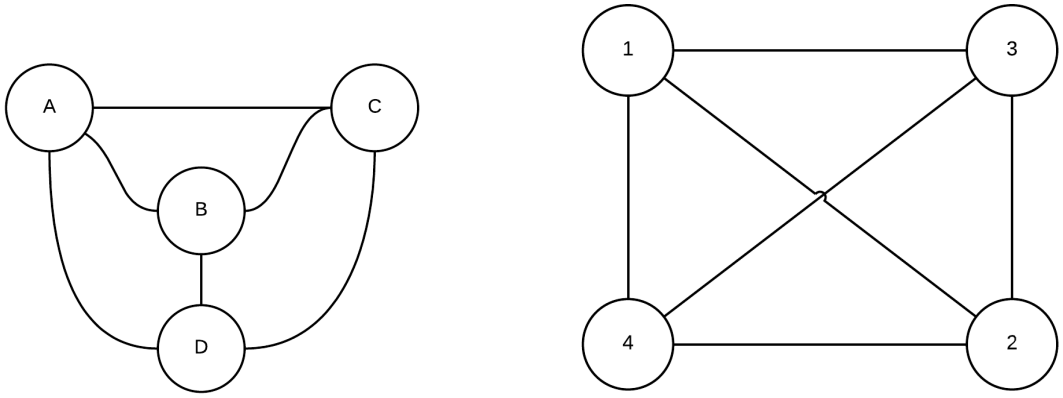


Figure 2.4: Sample graph.

There are numerous properties of graphs, of which, homomorphism is of particular interest for graph transformation systems. A homomorphism is a relationship between two graphs (or sub-graphs) that is similar to the notion of equivalence. Two graphs are considered homomorphic if a bijective relation exists between the vertex sets of the two graphs. Formally, for two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and a relation:

$$f : V(G_1) \rightarrow V(G_2) \tag{2.3}$$

Such that any vertices u and v in V_1 which are adjacent in G_1 have a mapping through f to two vertices a and b in V_2, and a and b are adjacent in G_2. Alternatively $(f(u), f(v)) \in E_2$. Consider Figure 2.5 which contains an homomorphic pair of graphs, these are related through the relation $f = \{(A, 1), (B, 2), (C, 3), (D, 4)\}$.



(a) G_1 (b) G_2

Figure 2.5: G_1 and G_2 are homomorphic graphs

Graph homomorphism is the core concept used in graph transformation systems to match transformation rules to the host graphs and to search the state space of a transformation systems for states at are semantically identical. A stronger version of homomorphism called isomorphism also exists and can be of use. A pair of graphs are isomorphic if for a homomorphism, f, its inverse f' is also a valid homomorphism.

Graphs can also contain information within each vertex or on each edge; this notion is formalized by concept of graph labeling. UML class diagrams are examples of labeled graphs where information about the class is expressed within each class node and the edges between class nodes are often labeled with multiplicity relationships. The concept of labeling is used in graph transformation to annotate nodes with values. More complex schemes and syntaxes have been developed which allow for annotations to encode a variety operations. Some of these are described in section 2.3.1 in the context of the GROOVE graph transformation tool.

Transformation

Transformation of graphs are accomplished through a set of graph transformation rules. These are also referred to as graph production rules since a graph transformation rule set can be considered a grammar for the production of graphs. Production rules operate on a host graph which is transformed by each rule as it is applied. Each production rule in a graph grammar contains a left hand side (LHS) and right hand side (RHS), similar to text-based grammars. If and when conditions imposed by the LHS of the grammar are met by the host graph (or a subgraph of the host graph), the rule is "fired" and the transformation on the host graph is completed. Consider the example presented in Figure 2.6 of a production rule which removes a node.

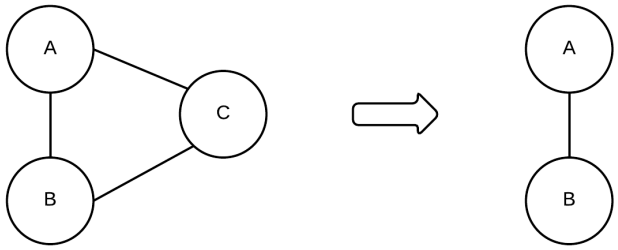


Figure 2.6: Sample graph production rule which removes a single node from a fully connected tri-subgraph

Figure 2.7 shows a host graph before and after the application of the rule described in Figure 2.6. The rule presented in this example can be interpreted two different ways. The first, the Double Push Out (DPO) approach would apply this rule *exactly* one way and would produce the transformation seen in Figure 2.7. The second interpretation, the Single Push Out (SPO), could be applied in three different ways, depending on how one assigns the nodes A, B, and C. In general, the DPO approach will not allow for "dangling" edges (edges without a node on one end) to exist after transformation is applied Alternatively the LHS of the rule must match exactly into the host graph (including edges and nodes). The SPO allows for rules to be applied that will generate dangling edges and simply deletes them after the rule is applied. GROOVE, the tool used for this project uses a SPO approach.

A formal definition of a graph transformation system, adapted from the definition in [4], is a set of production rules P which act upon a host graph G. Each rule $p_i \in P$ consists of a LHS and RHS, L_i and R_i respectively. The pre-conditions that must be

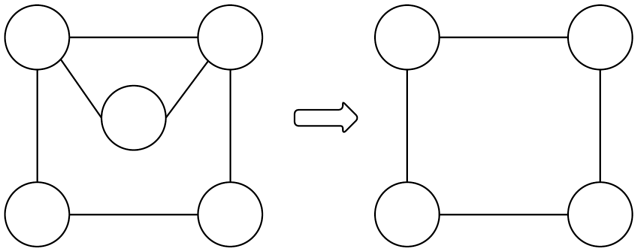


Figure 2.7: Sample host graph before and after rule application.

met within the host graph are given by the both the topology and labeling of a nodes and edges in L_i. R_i describes the state of the graph (or subgraph) after the rule p_i is applied. Transforming G to the resulting graph G' via production rule p_i using homomorphism o can be denoated by $G \Rightarrow_{p_i,o} G'$ such that the following are true:

- $o(L_i) \subseteq G$, the LHS has an homomorphic mapping to a subgraph of the host graph.
- $o(R_i) \subseteq G'$, the RHS has a homomorphic mapping to the subgraph of the resulting graph.
- $o(L_i \setminus R_i) = (G \setminus G')$ and $o(R_i \setminus L_i) = (G' \setminus G)$, the difference between the rules is equivalent to the difference between the host and and result graph.

Properties of Transformation Systems

Graph transformation systems have many properties that describe the state and execution of the transformation. Not all transformation systems are required to have these properties, however many applications of graph transformation may find these properties desirable. Those of interest for this work are:

- Termination
- Confluence
- Convergence
- Correctness

For a transformation system to satisfy the **termination** property it must eventually complete. Alternatively, it is not possible for an infinite sequence of rules to be applied [9].

For a transformation system to satisfy the **confluence** property all possible sequences of executing rules must result in the same final graph state. Formally, for graph states a, b, c, d if $b \leftarrow^* a \rightarrow^* c$ then $b \rightarrow^* a \leftarrow^* c$ [9].

Convergence is a property and holds if the transformation system is both terminating and confluent.

The property of **correctness** is not a general property of a graph transformation system, rather it is application specific; each application domain may impose specific constraints. For example, if a transformation system was operating on a graph describing bicycles it might be a domain constraint that all bicycles have exactly two wheels. Then after each application of a rule every bicycle should have two wheels for the correctness property hold.

2.3.1 GROOVE Transformation Tool

The GRaph based Object Orientation VErification (GROOVE)[3] tool chain is a set of Java libraries that supports creation, execution, and verification of graph transformation systems [40]. The GROOVE tool is based on an algebraic approach to graph transformation, and uses a single push-out transformation system [40]. GROOVE shares many features with other graph transformation systems such as AGG, FUJABA[4], PROGRES [2, 38] however, it also has a unique combination of features that make it very powerful and easy to use; these include a graphical interface for creating rules, optional graph typing, regular expressions, and model checking with several different exploration strategies [13]. The following aims to provide a brief introduction to GROOVE's rule description format, typing, and simulation; [13] provides a more comprehensive introduction to the tool through four examples. The remainder of this section on GROOVE will use a simplified prescription example. Note that these examples are not representative of the main work of this thesis.

[3] `http://groove.cs.utwente.nl/`

[4] http://www.fujaba.de/about-us.html

Typing

GROOVE supports typing that is similar to the concept of typing in object oriented programming. Figure 2.8 shows a sample type graph capturing the relationships between a medication and prescription. Regular arrows denote regular edge relationships between node types. Inheritance or extension is captured via an arrow with a solid white head, in Figure 2.8 this is between the "BrandName" and "Name" types, indicating that "BrandName" is a specialization of the "Name" type.

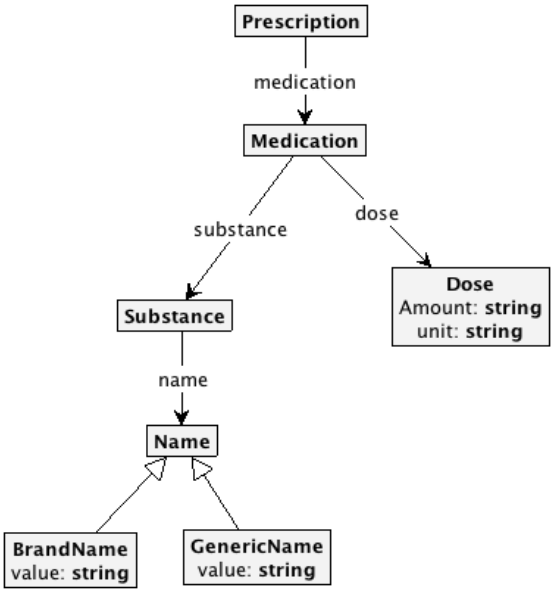
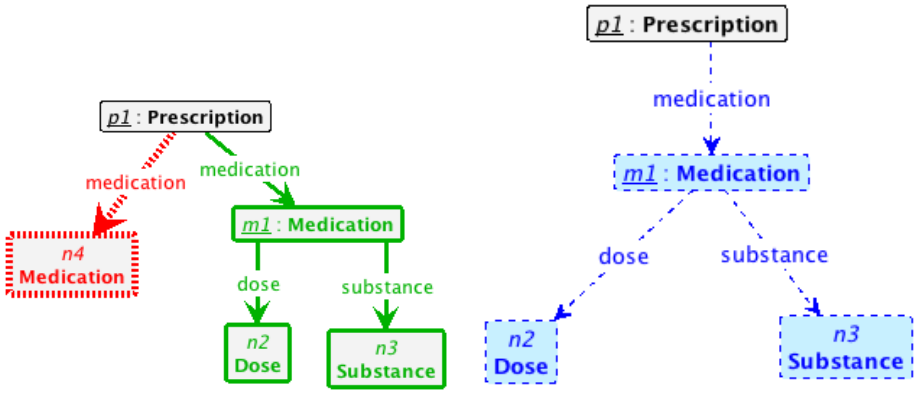


Figure 2.8: Sample type graph in GROOVE.

Transformation Rules

GROOVE represents transformation rules using an inline representation format where the left and right hand sides of the rules are combined together using different types of operations on the nodes and edges in the graphs. In Figure 2.9 two rules are shown. In Figure 2.9a the addition of a new "Medication" type object is performed, as indicated by the green arrows and node. Note that the embargo or "not" relationship between the Prescription and Medication node (node n4), this indicates that this rule can only be applied to a Prescription object that does not already have a Medication object. In 2.9b a Medication type node (and it's subnodes) are deleted as indicated by a deletion edges and nodes (in blue).



(a) Addition rule (b) Deletion rule

Figure 2.9: Sample addition and deletion rules in GROOVE.

In addition to adding and deleting nodes and edges, GROOVE also supports a limited set of arithmetic and string operations. These can be used to control rule application by specifying conditions that refer to the values that annotate the nodes of a graph. Consider Figure 2.10 which adds 10 mg to the dose of a medication, but only if the dose amount is less than 100 mg and the unit for the dose is "mg".

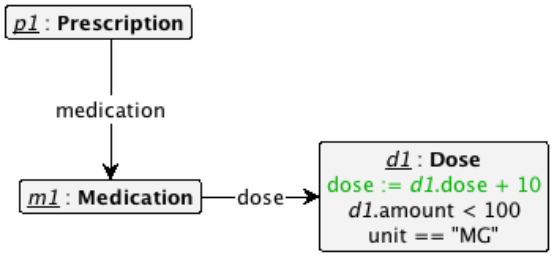


Figure 2.10: Manipulation of element annotations.

GROOVE contains many more advanced features for annotating rules and for checking conditions within graphs. This short description does not do the tool justice. Rather it was meant to provide the reader with enough background to allow them to understand descriptions of rules throughout the rest of this work.

Control Flow

Since graph transformation systems are non-deterministic (meaning rules are applied in no particular order) GROOVE provides a mechanisms to specify when particular rules can be applied. This helps reduce the complexity of transformation systems

and assists with modeling systems that are inherently deterministic and would be cumbersome to model in a non-deterministic manner. This functionality comes in the form of both rule application priority levels, and simple control programs. Each transformation rule (like those presented in the proceeding section) can be assigned a priority; higher priority rules are always executed when possible before lower priority rules that also match the current host graph. Control programs allow for the user to specify precisely the order in which to attempt to apply rules. Control programs are written in a domain specific language that is similar to the C programming language. A sample control programming is shown in the code listing below.

```
until (stop) {main();}

function main(){
    while(true){
        try{addDose;}
        else try{addMedication;}
        else {deleteMedication;}
    }
}
```

Model Checking

GROOVE provides a model checking tool set that will explore the state space of the transformation system. The state space is all possible states that can be generated by the transformation system. A variety of exploration strategies are available, including Breadth First Search, Depth First Search, and Linear Exploration. The model checking tools are capable of checking the entire state space for whether certain conditions hold. For example, one might wish to know that the dose of medication will never exceed 100 mg, this can easily be captured in a conditional expression that are then checked in each state during state space exploration. The GROOVE user interface also provides a tool for visualizing the state space and exploring it manually. These tools are invaluable while debugging transformation systems. Using these tools, one can be assured that particular properties of a transformation system will hold for a particular input. For example, the confluence property of the system can be checked for a specific start graph by generating the entire state space and then checking that all possible execution paths result in the same final state. Note that the model check-

ing facilities provided by GROOVE do not constitute proofs that properties hold in the general case.

2.4 Domain Specific Languages

A Domain Specific Language (DSL) in the context of computer systems, is a programming language that has been tailored to a very specific domain or application. This is in contrast to a General Purpose Language (GPL) which has a much broader set of language concepts suitable for programming in many different application domains. DSLs are ideal for application areas that require user input, but do not need the full rigour of a general purpose language, and/or need a less complex input format. In general, DSLs allow their users to express concepts from an application more concisely, but sacrifice the transferability of the language to other domains [28].

DSLs are in wide use in many computer systems, both for software development and end-users. Well known DSLs include HTML, the language used to structure content for web pages; SQL, for creating database queries; Mircosoft's Excel spreadsheet formula language; CSS, used specify styles for web page components; and LaTex, used to typeset documents (including this one) [28].

DSLs can be extensions of existing general purpose languages where the DSL leverages and adds to the existing syntax and language environment to provide additional expressiveness for a particular application. An example of such a DSL would be AspectJ[5] which is tool for supporting Aspect Oriented Programming (AOP) in Java. AspectJ provides annotations that can be embedded in existing Java code and are addressed during compilation or runtime.

Other DSLs are completely independent of general purpose languages. For example, CSS (Cascading Style Sheets), is a language used to describe the formatting and appearance of elements on a web page. This language, though parse-able by any number GPLs is not itself built upon any specific language. Many DSLs are hybrids which utilize a minimal set of features from other languages and also define their own syntax and semantics.

[5]https://eclipse.org/aspectj/

2.4.1 ANTLR

ANother Tool for Language Recognition[6] (ANTLR), is a Java based tool for generating parsers for user-defined languages [34]. ANTLR is widely used in both academia and industry with notable users being Oracle, Twitter, and NetBeans [34]. From a theoretical perspective, ANTLR uses a LL(*) parsing strategy introduced by Parr (also the creator of ANTLR) in [35]. The fundamental idea of LL(*) parsers is to use regular expressions to match non-terminal characters in an input string rather than using traditional techniques such as backtracking [35].

The ANTLR tool takes a modified Backus-Naur Form (BNF) of the language as input. The input format also supports Java annotations that can be executed by the parser. The tool produces a series of Java classes that can then be used to parse the desired input. The three main classes produced are a lexer (lexical analyzer), a parser, and a listener. The lexer is used to produce meaningful tokens from the input character stream and the parser generates a parse tree for the tokens produced by the lexer. The listener is a class used to assist programmers in working with the resulting parse tree; it provides a set of methods that can be called to walk the parse tree and perform the required application specific computations [34].

ANTLR (now at version 4, denoted ANTLR4) was the tool used during this project to produce the DSL for prescribing. Chapter 3 describes the DSL in detail.

[6] http://www.antlr.org/

Chapter 3

Domain Specific Language

For the purpose of this project, a domain specific language (DSL) was developed to provide an interface for users to input prescription information into a hypothetical e-prescription system. The goal of this language was to find a balance of expressiveness and functionality. Prescriptions have many concepts that are traditionally captured and communicated using natural language, these concepts were described in Chapter 2. The DSL that was developed attempts to closely mimic natural language expression while still providing structure. This chapter details the DSL and provides examples for use.

3.1 Design

Mernik *et al.* have described a process and a set of patterns for DSL development [28]; this process involves the following:

1. **Decision** to use a DSL
2. **Analysis** of the domain
3. **Design** of the DSL
4. **Implementation** of the DSL
5. **Deployment** of the DSL into the application domain.

This process, and its associated patterns, was used to guide the development of the DSL for prescribing. Patterns for DSL development tend to describe a commonly

occurring problem or situation, and can be used to make suggestions about how to proceed in development. The remainder of this section describes each of the aforementioned stages of development and discusses how the patterns identified by Mernik *et al.* were used to support the design of the prescribing DSL.

Decision

The decision to undertake DSL development requires there be an existing need for change in the way some technology is used [28]. As previous chapters have illustrated, there is a need for some other form of specifying prescriptions in electronic form. DSLs appear to be one viable option that could capture the complex nature of prescriptions. Mernik *et al.* describe several patterns that can be used to make decisions regarding whether or not to develop a DSL, those that fit within the scope of this work are:

- Changing user **Interaction** with a system
- **Domain Specific Analysis, Verification, Optimization, Parallelization, and Transformation**(AVOPT) of data and its processing
- **Data Structure Representation** of input data

The first pattern, changing user interaction, is the most readily applicable pattern. One of the main goals of this work it to provide a new method of entering prescription information, this directly supports the application of this pattern. The second pattern, AVOPT, indirectly applies. The AVOPT pattern would be supported by any method of input that facilitates AVOPT. The third pattern, data structure representation, was also identified. A prescription is a type of data structure; as will be seen in Chapter 4, a DSL could be used to identify the components of this data structure. Other patterns presented by Mernik *et al.* apply indirectly to the domain of e-prescribing. The applicability of three of the patterns supports the decision to develop a DSL for e-prescribing.

Analysis

The analysis stage of DSL development focuses on understanding the application domain [28], this includes the following:

- Determining the **scope** of the domain

- Understanding terminology and **syntax** of the domain
- Understanding the **semantics** of syntax in the domain
- Understanding how the syntax and semantics are used to specify domain concepts (**pragmatics**).

Mernik *et al.* identified three patterns for analyzing a domain: i) formal analysis, ii) informal analysis, and iii) extraction from existing code. Of these three approaches, the second, *informal analysis* was used. The informal analysis consisted of examining several examples of prescriptions from a number of sources, including a domain expert. Subsequent discussion with a domain expert determined which concepts of prescribing were the most important and how these concepts could be expressed most effectively. One of the important parts of the analysis was understanding how those who prescribe medications "think" about prescriptions. This was important for determining how the language elements of the DSL could be combined together; alternatively this describes the pragmatics of the language. For example, it was valuable to understand that while the dose of a prescription can be expressed in a number of "tablets" of a medication, the quantity that the prescriber may think about is the absolute amount of active ingredient (10 mg, 2 mL etc.). Therefore the DSL should be able to describe an absolute amount of substance rather than relying on a number of tablets for specifying the dose of a medication.

During this analysis phase,the scope of the DSL was also determined. Prescribing occurs in a number of unique clinical settings; the types of prescriptions in a hospital oncology unit will differ dramatically from prescriptions written in the office of a primary care physician. Each of these settings has specific requirements in terms of prescribing concepts that are relevant. For example, a prescribing DSL for a hospital ward would need to have language constructs for describing drip rates of intravenous medications. However, the concept of IV drip rate is significantly less important in family practice where the majority of prescriptions are written for substances that are taken orally or topically. The scope of this DSL was determined to be a subset of the prescribing concepts commonly found in primary care.

The analysis portion of DSL development was tightly coupled with the design and implementation components. These three phases were conducted in an iterative manner with each iteration adding a new prescribing concept to the DSL.

Design

Mernik *et al.* describes DSL design as having two factors i) a relationship to existing languages, and ii) a language description [28]. The first factor can be one of two design patterns, either *language exploitation* or *language invention*. The prescribing DSL utilized both of these patterns. Prescribers have an informal language they use to communicate prescription information. Since the DSL in this project is based on this informal language, the pattern of *language exploitation* fits. However, this use of language exploitation is different than Mernik *et al.*'s original description in the sense that it does not exploit a pre-existing programmatic language. The pattern of *language invention* also applies, as the language features, though derived from an existing informal prescribing language, are unique in their structure.

The second factor, language description, can either be *formal* or *informal*. Since this language was described using a grammar, and is tied to a graph transformation system that describes the prescription at a semantic level (Chapter 4), it is a formally described language. It should be noted that the language is actually a *partial* formal language, this is discussed in detail in Chapter 6.

The most important concept that was developed during the design phase of DSL development was how the core concepts in a prescription should be described. It was determined that a prescription should have four main syntactic components. These components are related via the semantics and pragmatics of the language to the core components of a prescription as described in Chapter 4. When combined, components together make up a prescription "atom". The components are as follows:

1. Action
2. Medication
3. Dose
4. Timing

A result of this design is that all DSL features related to describing a prescription must fall within one of these four categories. It was also determined that several prescription's "atoms" could be able to be combined together to express a more complex prescription. This was accomplished by using a set of keywords to specify the semantics of the relationships between the prescription atoms. For example,

the keyword “THEN” can be used to indicate a temporal relationship between two prescription atoms: “atom x THEN atom y”.

Implementation

Since the prescribing DSL is used to produce an input data structure for a graph transformation system (described in Chapter 4), it was considered to fit the *pre-processor* implementation pattern [28]. Pre-processor type languages are characterized by using simple lexical and parsing approaches to translate input to a different language or model for later processing. In the case of the prescribing DSL, the model to translate to is the input model for the graph transformation system.

ANTLR4 (ANother Tool for Language Recognition v4) [34] is a Java based tool for parsing user defined languages. This tool parses a BNF style grammar for a language and generates a number of utilities for parsing that language. This tool was an excellent fit for the prescribing language as its utilities created the right amount of functionality for a pre-processor type language implementation. Prior to ANTLR4 other tools, including XText and TXL, were experimented with. Though no formal comparision of tools was made, ANTLR4 was found to be the best fit for the application based on experimentation.

The main utilities that ANTLR4 provides to the DSL developer are: i) a lexer ii) a parser iii) a parse tree walker. Figure 3.1 shows these utilities linked together as a pipeline to produce the Rx Graph Model, which is the input to the graph transformation system described in Chapter 4.

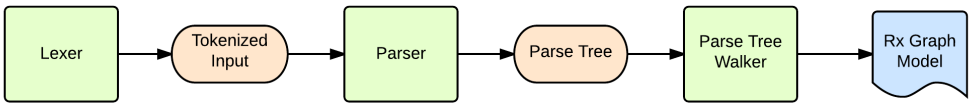


Figure 3.1: Pipeline for parsing of DSL to produce the Rx Graph Model.

The parse tree walker utility generated by ANTLR4 provides an abstract class called a “Listener” for interacting with the parse tree. This Listener class can be customized (via sub-classing) by the developer. As the tree is walked, the functions of the Listener class are called. Using a Listener the Rx Graph Model was generated from the parse tree.

In addition to a pipeline that makes up the back-end of the DSL, a simple graphical

user interface (GUI) was created using Java's Swing environment. This interface provides a simple text field for writing prescriptions. It supports the DSL by providing simple syntax highlighting for keywords. This GUI also triggers the application of the graph transformation system (described in Chapter 4). A screenshot of the GUI can be seen in Figure 3.2.

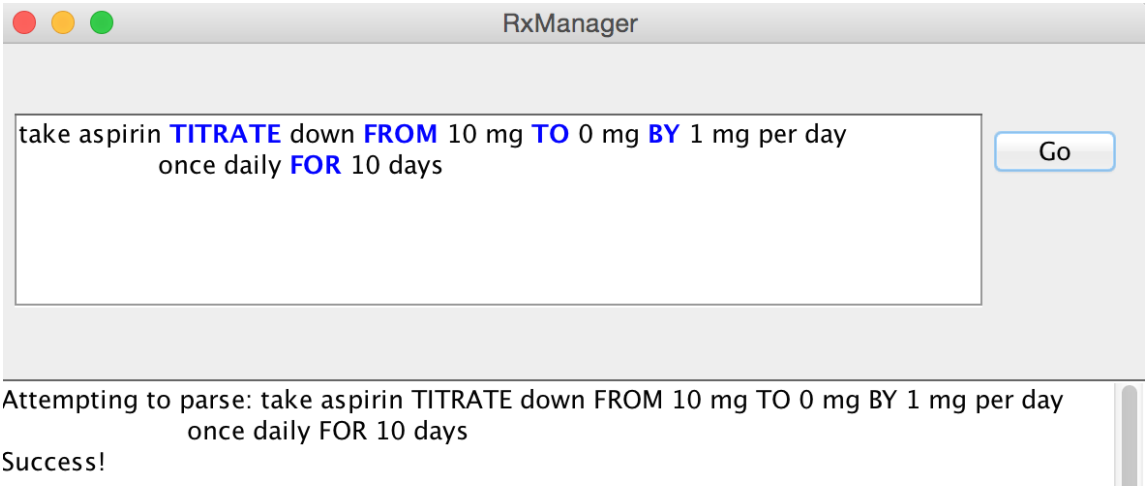


Figure 3.2: Simple GUI for writing prescriptions in the DSL.

3.2 Syntax

The syntax of the DSL was specified using an ANTLR4 grammar, which is similar to the BNF of a grammar. The full grammar is visible in Appendix A. The following describes a selection of core syntactic constructs of the DSL.

As discussed in previous sections in this chapter, the main assumption in the DSL is that prescriptions can be expressed in terms of four components: an action, a medication, a dose, and timing. This is reflected in the syntax by each prescription atom having those four components (in that order):

```
atom : action medication dose timing repeat?;
```

The last component in expression directly above is a "repeat" component. This is a language feature that supports the concept of repeating the same prescription in time. Atoms are part of an expression that can be used to link several of them together:

```
expr : expr 'THEN' expr | atom | expr NEWLINE ;
```

3.2.1 Examples

To show the types of prescriptions concepts that can be expressed by the prescribing DSL, a number of examples are given below.

Example I: Basic

This example shows a basic use of the core elements of the DSL.

```
take Aspirin 81 mg once daily FOR 10 days
```

The components of this prescription are assigned as follows:

- **Action:** "take"
- **Medication:** "Aspirin"
- **Dose:** "81 mg"
- **Timing:** "once daily FOR 10 days"

Example II: Specific Dose and Time

It is desirable to specify the exact time (hour, day of week, day of month) to and the dose. This is done by using a tuple to replace (or in addition to) the existing components.

```
take Aspirin (81 mg, 162 mg) twice daily (8, 20) FOR 10 days
```

In this example 81 mg of aspirin should be taken at the 8th hour of the day (8 AM), and 162 mg of aspirin should be taken at the 20th hour (8PM). The components of this prescription are assigned as follows:

- **Action:** "take"
- **Medication:** "Aspirin"
- **Dose:** "(81 mg, 162 mg)"
- **Timing:** "twice daily (8, 20) FOR 10 days"

Example III: Sequencing Prescriptions

As mentioned previously, several prescriptions can be strung together to create a sequence of instructions; this is accomplished by using the "THEN" keyword.

```
take Aspirin 81 mg once daily FOR 10 days
    THEN take Aspirin 162 mg once daily FOR 10 days
```

- **Action 1:** "take"
- **Medication 1:** "Aspirin"
- **Dose 1:** "81 mg"
- **Timing 1:** "once daily FOR 10 days"
- **Action 2:** "take"
- **Medication 2:** "Aspirin"
- **Dose 2:** "162 mg"
- **Timing 2:** "once daily FOR 10 days"

Example IV: Titrating Prescriptions

One of the features requested by the domain expert was the ability to change the dose of a medication over time by a set amount. This is the concept of a titrating or tapering prescription. This was accomplished by replacing the dose component of the prescription atom with a syntax to specify a dose of medication changing with respect to time.

```
take Aspirin TITRATE down FROM 100 mg TO 0 mg
    BY 10 mg per day once daily FOR 10 days
```

This describes a prescription for 10 mg of aspirin less every day. The keyword "TITRATE" describes a changing dose in time, in this case the "down" indicates the direction to change in, alternatively "up" could be specified to show an increase in dose over time.

- **Action 1:** "take"

- **Medication 1:** "Aspirin"
- **Dose 1:** "TITRATE down FROM 100 mg TO 0 mg BY 10 mg per day"
- **Timing 1:** "once daily FOR 10 days"

3.3 Summary

In this chapter a Domain Specific Language (DSL) for authoring prescriptions was presented. The syntax of this DSL has been created to make the language easy to learn and use for clinicians. This language is a prototype and has yet to be evaluated for usability. A design process for creating a DSL described by Mernik *et al.* was used to support the creation of the DSL. This design process had five steps, decision, analysis, design, implementation, and deployment; four of five of these steps were carried out over the course of this project (deployment did not occur). To implement the DSL, a Java based tool called ANTLR which produces utilities for parsing textual input, was used. A discussion of the DSL's strengths, weaknesses, and next steps can be found in Chapter 6.

Chapter 4

Graph Models

Prescriptions authored in the Domain Specific Language (DSL) presented in Chapter 3 can be parsed and expressed as graphs. Expressing prescriptions as graphs provides a way to formally specify the semantics of the prescription and/or the languages. More specifically, these semantics are described using a graph transformation system which consumes an input graph model (Rx Graph Model) and produces output graph model (CoreRx Graph Model). This process is captured by Figure 4.1. This chapter presents both graph models and the rationale for their design; then the transformation system is presented.

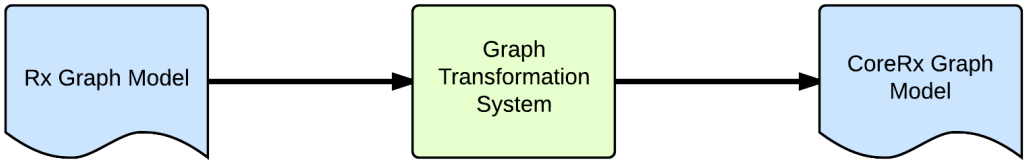


Figure 4.1: High level description of graph model transformation.

4.1 Rx Graph Model

As described above, the Rx Graph Model is the input model to the graph transformation system. It provides an expression of the prescription in a graph format. A GROOVE type graph of the Rx Graph Model is represented in Figure 4.2.

The main components of the Rx Graph Model are as follows:

- A **Prescription** for a medication including a start date, number of repeats, and a duration.

- A **TimeFrame** to describe the timing of the prescription. The duration of the prescription is expressed in number of consecutive TimeFrames.
- A set of **TimePoint**s which indicate the points of time within the respective TimeFrame.
- A **Dosing** associated with a TimePoint which specifies the dose (amount and unit) of the medication.
- **Titrate** which, if present, describes how the Dosing element should change in time.

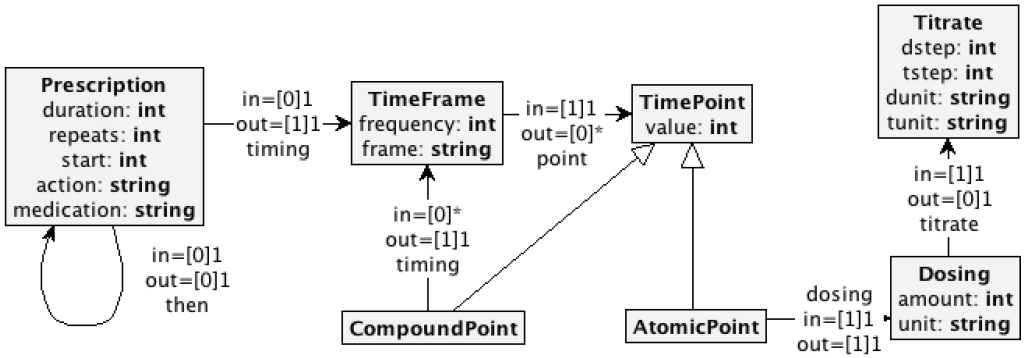


Figure 4.2: Atomic prescribed medication action data model.

The duration field in the Prescription component describes the number of times to repeat the indicated TimeFrame. This differs from the concept of "repeats" (described by the repeats field) which specifies how many durations to repeat. The "then" edge that may connect two Prescriptions shows a temporal relationship between two distinct Prescription objects (prescription A THEN prescription B); this is analogous to the concept of "THEN" presented in Chapter 3. The "start" field of the Prescription node describes a time (given as number of seconds since the UNIX epoch, January 1st 1970) the Prescription should start.

As seen in Figure 4.2, TimePoints can be either AtomicPoints or CompoundPoints. A CompoundPoint can be further refined by adding additional time frames that specify the timing within the interval defined by the granularity of the TimePoint in question. This structuring of CompoundPoints and TimeFrames is similar to recursion where each level of recursion provides a more detailed specification of the timing. AtomicPoints are "atomic" in the sense that they can be no longer expanded,

and must have a dosing associated with them. AtomicPoints serve as a base case for the recursion of the TimeFrame/CompoundPoints. Using this recursive structure, arbitrarily complex timings can be described to any level granularity (provided there are appropriate units for the TimeFrame). For an example of timing, consider this simple prescription given in natural language (nesting TimeFrames is not supported by the DSL):

```
take warfarin 10 mg once every Monday at 2 PM for 8 weeks.
```

Which is represented in the Rx Graph Model as:

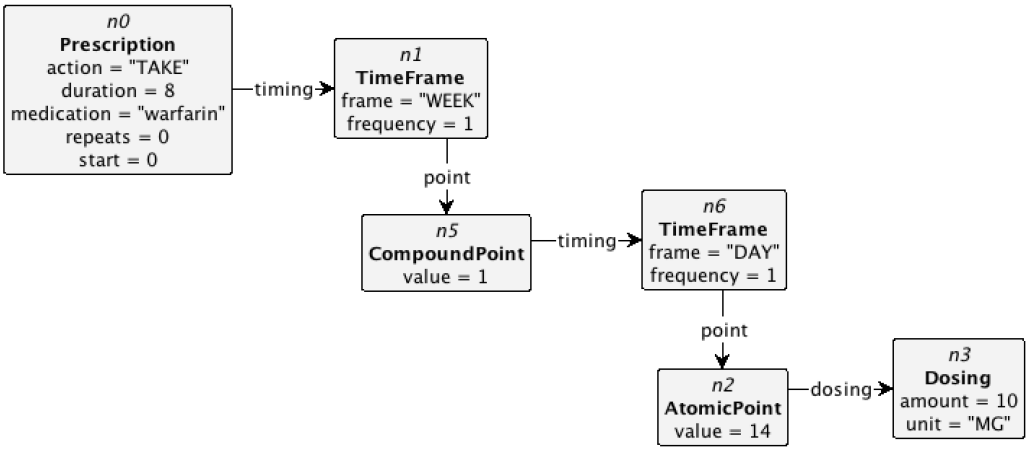


Figure 4.3: Using TimeFrame and TimePoints to specify a particular time.

As previously mentioned, the Titrate component of the the Rx Graph Model specifies how the dose changes over the course of the prescription plan. The "dstep" and "dunit" indicate how much to change the dose by for each occurrence of the "tstep" and "tunit". For example, the Titrate component of a prescription might specify "dstep" = -10, "dunit"="MG", "tstep"="2", and "tunit"="DAY", which would result in a prescription that decreases the dose of the medication every two days.

As an example a titrating prescription, consider this prescription given in the DSL:

```
take warfarin TITRATE down FROM 10 mg TO 5 mg
BY 1 mg per day once daily FOR 5 days
```

which can be represented in the Rx Graph Model as seen in Figure 4.4.

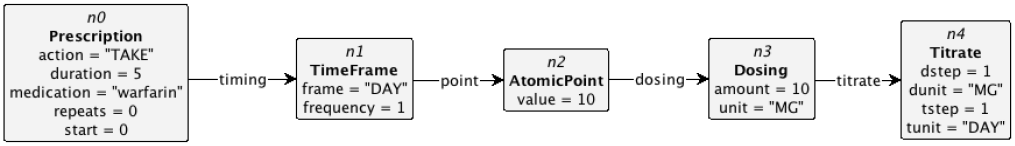


Figure 4.4: Example of Rx Graph Model with a titrating prescription.

4.2 CoreRx Graph Model

The coreRx Graph Model for prescribing represents one of the key novel aspects of this work. The central concept of the core model is that every prescription can be represented as set of individual medication events. Each of these is considered an Atomic Prescribed Medication Action (APMA), which is more precisely defined as:

> *The information that is required to specify an action related to a dose of medication at a particular instant in time. Such that if any one piece of information is removed, then the APMA is no longer captures the intent of the prescription.*

There are four main components of an APMA:

- A **medication** which describes the substance (name, form, route) to be taken.
- A **dose** which specifies how much substance should be taken.
- An **action** which indicates what to do with the **medication** in question.
- A **timing** which specifies an interval in time at which to take the **action** with respect to the **medication**.

Figure 4.5 shows the four main components of an APMA in a conceptual model. *Timing* is expressed as two integers, start and stop, which specify the time interval in seconds from the UNIX epoch (January 1st 1970 UTC). It is worth noting that an interval in time, as required by an APMA for timing, can also represent an "instant" or "point" in time if both the start and stop are the same.

The *action* field of an APMA provides a significant amount flexibility in specifying the prescription. The most trivial action "TAKE" is a catch all action that implies that the patient should consume (apply, inject, etc...) the medication in question. Negative actions can also be given such as "DO NOT TAKE" which indicates that the

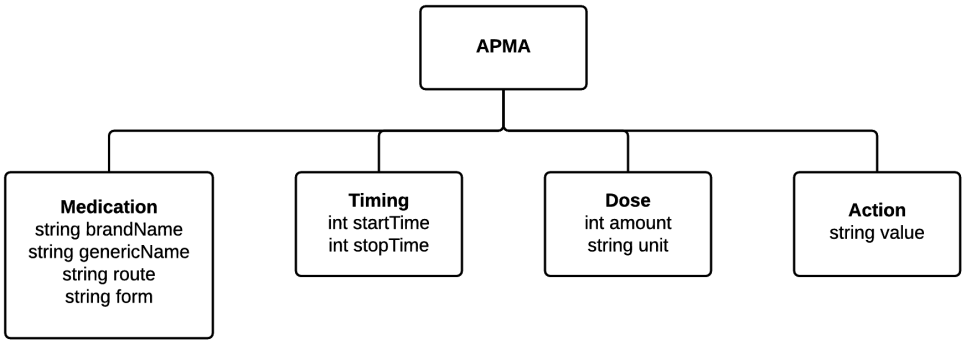


Figure 4.5: Atomic prescribed medication action data model.

patient should not undertake the action described by the APMA. Another negative action is "TAKE NO MORE THAN" which indicates that the patient should not exceed a dose in the specified time period. The semantics of these actions are roughly equivalent to their natural language interpretation and will be precisely defined in Chapter 5.

The coreRx Graph Model can be created by generating a collection of APMAs which describe actions for every interval in time for the prescription. This model consists of a "plan" graph node with edges to APMAs for every interval of time for that prescription. The type graph used by GROOVE for coreRx Graph Model is in Figure 4.6.

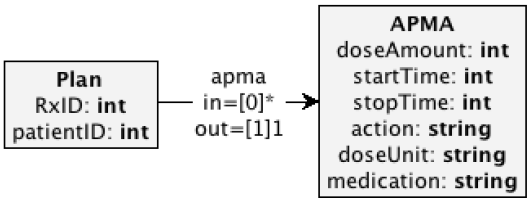


Figure 4.6: GROOVE type graph for coreRx Graph Model.

Figure 4.7 shows an example of a prescription specified in the coreRx Graph Model which was produced via the graph transformation system being run on an Rx Graph Model. In this example coreRx Graph Model,all time intervals have been specified with either a "TAKE" or "TAKE NO MORE THAN" action; this provides a complete specification for the prescription.

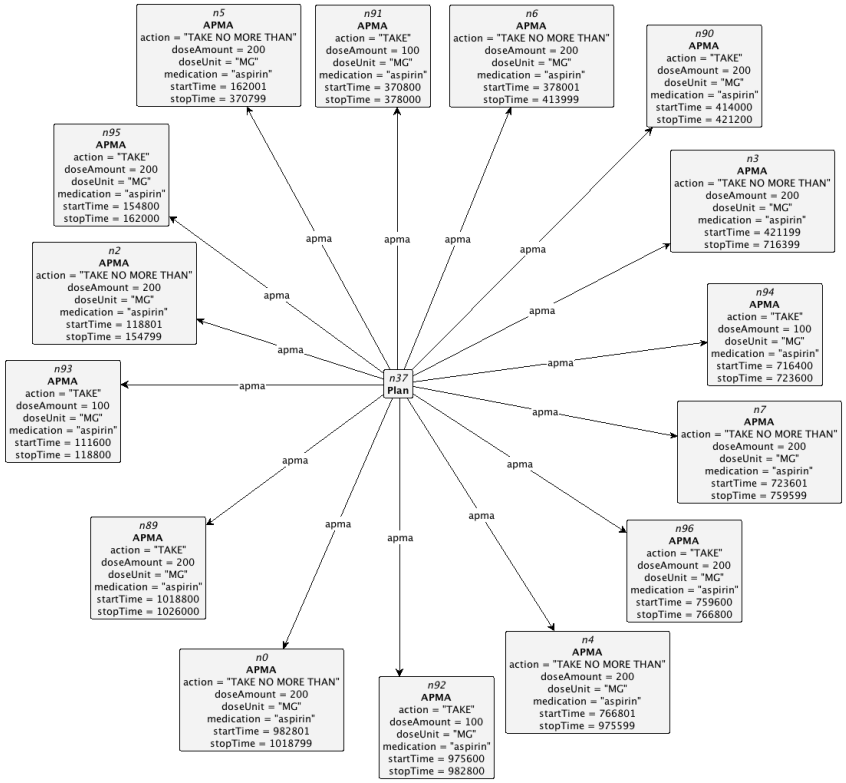


Figure 4.7: GROOVE type graph for coreRx Graph Model.

4.3 Model Transformation

To generate the coreRx Graph Model from the Rx Graph Model, a graph transformation system was used. The transformation was broken into several functional steps:

1. **static semantic validation**
2. **time unrolling**
3. **temporal unsharpening**
4. **plan completion**

These steps were captured via GROOVE's control program functionality. The full control program is visible in Appendix B. The following sections describe each of these steps in detail and provide examples of rules and conditions where appropriate. The entire transformation system ruleset can be found in B.

4.3.1 Static Semantic Validation

In traditional programming languages, a compiler is able to perform semantic checks of the input to determine if it violates any rules of the programming language. This is often done by examining abstract syntax graphs of the input. One of the benefits of specifying a graph model and transformation system for the prescribing DSL is the ability to check the semantics of input. This is important because, given only the Rx Graph Model, it is possible to specify a prescription that is not semantically sound. These semantic violations may include mismatching frequencies, incompatible units, missing components, or incorrectly combined components.

The GROOVE type graph system provides *some* checking for these violations, specifically related to missing or incorrectly combined components. For example, given the type graph for the Rx Graph Model presented in Figure 4.2, the graph in Figure 4.8 below would be considered semantically incorrect due to the *Dosing* and *CompoundPoint*s being incorrectly combined. Indeed, GROOVE's type checking system has flagged this as incorrect (seen as red shading).

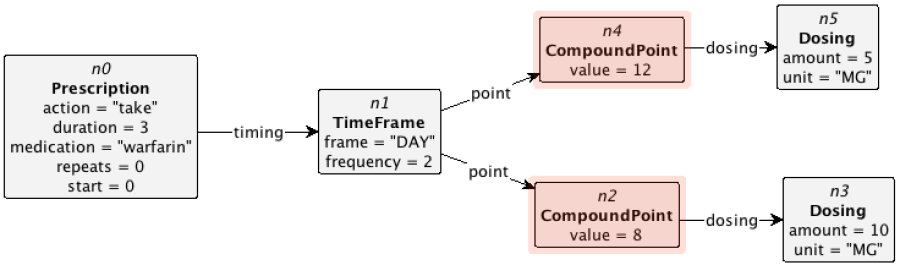


Figure 4.8: Semantically invalid Rx Graph Model due to poorly combined components

However, some semantics cannot be checked by GROOVE's built-in type checking system. As a result, a set of transformation conditions and rules were created to check for these problems. The most notable example is mis-matching frequencies, in which the frequency specified in a *TimeFrame* component does not match the number of connected *TimePoints*. Figure 4.9 shows an example of an Rx Graph Model with frequency mismatching.

A GROOVE condition that checks for frequency mismatching is shown in Figure 4.10. If this condition is met during the static semantic validation step of transformation, the entire transformation system enters an error graph state which prevents further computation.

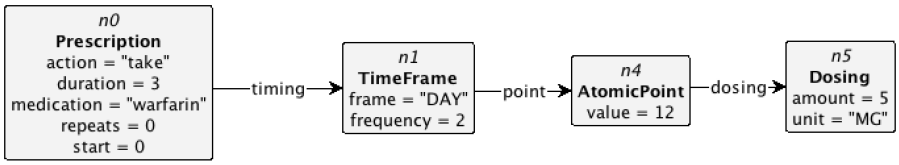


Figure 4.9: Semantically invalid Rx Graph Model due to frequency mismatch.

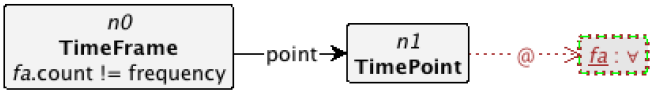


Figure 4.10: Static semantic condition for checking frequency mismatches on input Rx Graph Models.

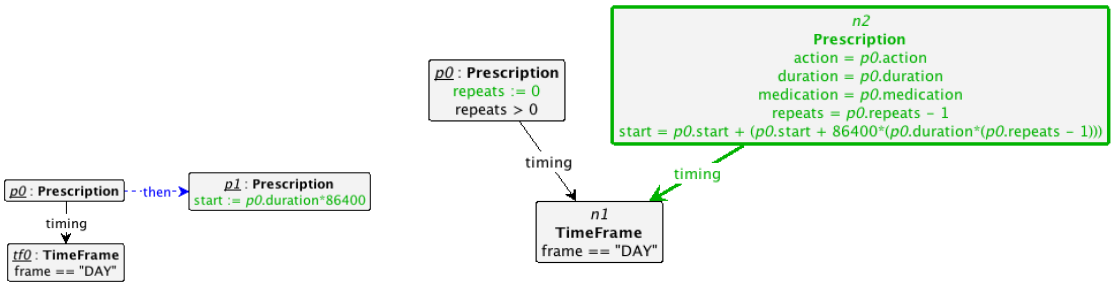
4.3.2 Time Unrolling

The time unrolling stage of the transformation is the most complex step; it is responsible for generating the set of APMAs that describe the prescription. Time unrolling is comprised of the following sub-steps:

1. Accounting for "then" edges between *Prescription* component
2. Accounting for "repeats" in the *Prescription* component
3. Accounting for "duration" in the *Prescription* component
4. Recursive expansion of *TimeFrame* and *TimePoint* components

"then" edges between *Prescriptions* are accounted for by adding the duration (in seconds) of the first prescription, *p0*, to the start of the second prescription, *p1*. Similarly, the concept of "repeats" in the Rx Graph Model is accounted for by duplicating the original prescription, *p0*, and adjusting the new prescription's start time. An example of both of these concepts is shown in Figures 4.11a and 4.11b respectively.

Similar to how "repeats" are handled, the concept of "duration" is accounted for by repeatedly duplicating the original prescription, *p0* until the entire duration has been accounted for. This expansion is similar to the notion of a "while" loop in traditional programming languages; in this case the condition for execution of this loop is "duration>1" and "duration" is decremented with every loop execution. An



(a) (b)

Figure 4.11: GROOVE rules to account for "then" and "repeat" concepts in the Rx Graph Model.

example of the rule that accounts for the duration of the prescription is visible in Figure 4.12.

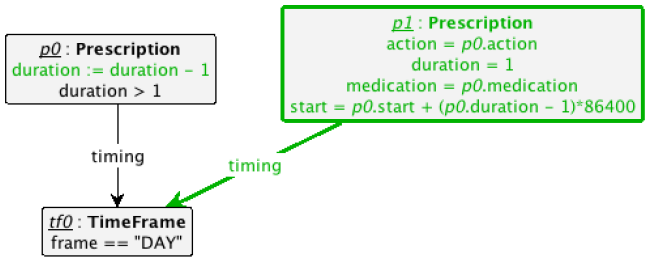
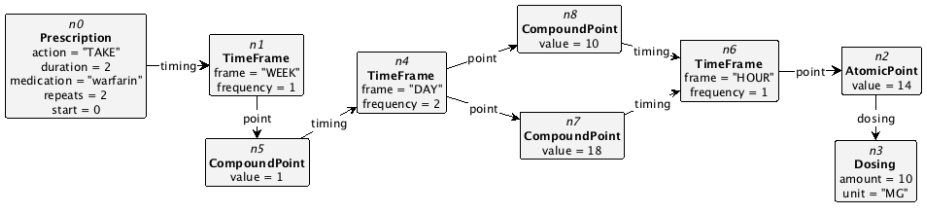


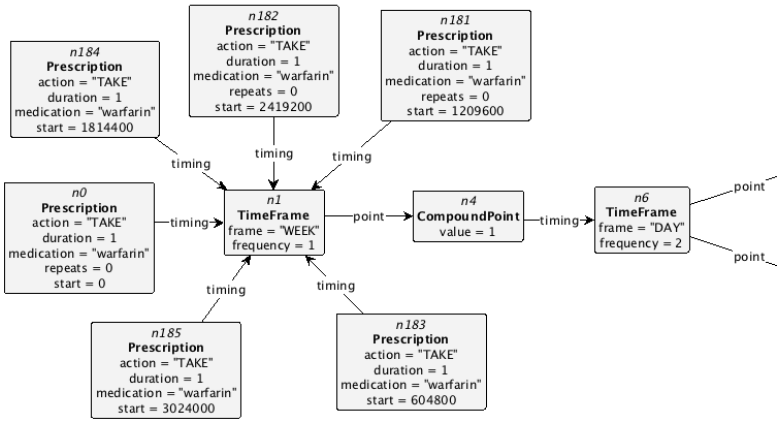
Figure 4.12: GROOVE rule to expand the duration concept of the Rx Graph Model.

After the application of the first 3 sub-steps of time unrolling, it is expected that a number of *Prescription* objects will be pointing to a single *TimeFrame*. Figure 4.13 shows an example of an Rx Graph Model before and after application of these first 3 sub-steps. All that remains in time unrolling is to recursively expand the remaining timing components.

As previously mentioned, expansion of the timing components is accomplished by a recursive algorithm. This algorithm's rule set works "top-down" in the sense that it starts at the *TimeFrame* that all of the *Prescription* objects are connected to, which is the least specific TimeFrame ("WEEK" in 4.13b). It continues until it reaches the base case of recursion, namely an *AtomicPoint*. A sample of a recursive rule and base case rule are shown in Figure 4.14.



(a) Before applying the first three sub-steps of time unrolling.



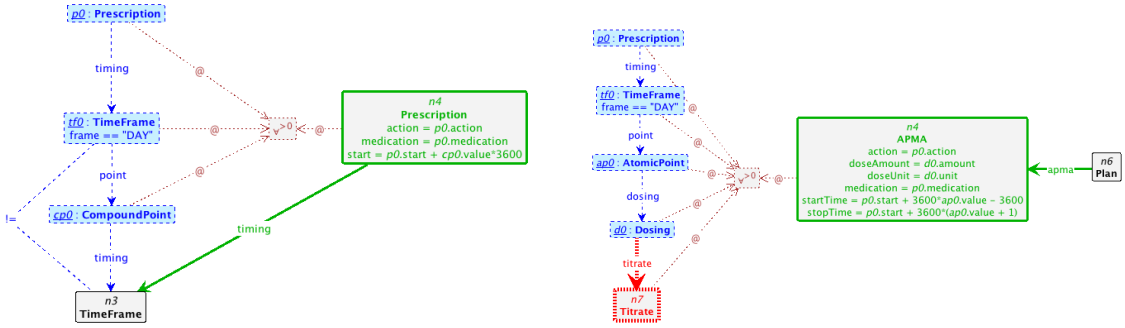
(b) After applying the first three sub-steps of time unrolling, remainder of timing omitted due to space.

Figure 4.13: Before and After accounting for "repeat" and "duration" concepts in time unrolling.

Continuing the example started in Figure 4.13, Figure 4.15 shows the expansion of the timing components.

4.3.3 Temporal Unsharpening

The purpose of temporal unsharpening is to change a the timing of the APMAs (generated in the previous steps) from instants in time to intervals. This is required because the transformation to this point specifies the APMAs with a degree of precision that is unreasonable for the majority of medications. For example, whether one takes their morning pill for hypertension at 8:00.00 AM or 8:00.01 AM is likely irrelevant, and one would not likely accuse somone of being non-adherent to their medication if they missed their scheduled dose by 1 second. Perhaps it only matters that it is taken within an hour of 8:00.00 AM. To accommodate this "fuzziess", an interval is generated in the temporal unsharpening step. In the current transformation



(a) Recursive rule. (b) Base case.

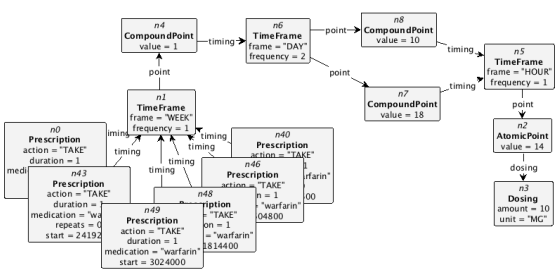
Figure 4.14: Recursive and base case rules for expanding the timing concepts of the Rx Graph Model.

system, the interval length is based on the highest resolution unit of time specified for the prescription. For example, if the prescription was specified to the resolution of an "HOUR" then the interval generated would be -1 hour and +1 hour from the scheduled time.

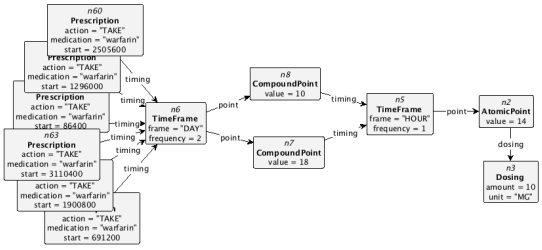
Though temporal unsharpening is conceptually a distinct step in the transformation process, it was combined with the recursive base case in this implementation of the transformation system. The result of this is visible in Figure 4.15d, where both a start and stop time are given for each APMA. This is done to reduce the number of rules that must be applied to generate the coreRx Graph Model, which reduces the state-space of the the transformation.

4.3.4 Plan Completion

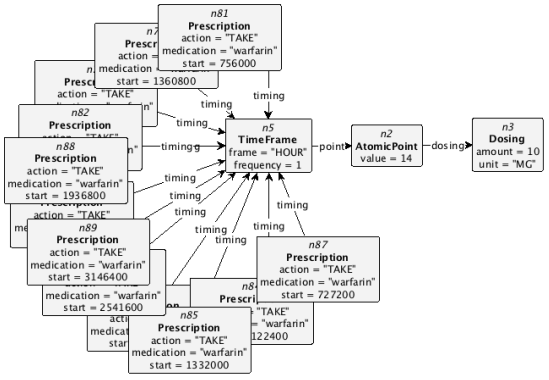
Part of the coreRx Graph Model presented above is the specification of all periods of time for a given prescription. The graph transformation system described thus far does not generate actions for intervals in time that are the "negative" (like the negative of a photograph) of the prescription actions. More concretely, if the coreRx Graph Model specifies a "TAKE" action for two intervals (see Figure 4.16a), three more intervals must be generated specify actions for the intervals before, between, and after. The process of "filling in the gaps" or generating the "negative" is called plan completion and a before and after example is shown in Figure 4.16.



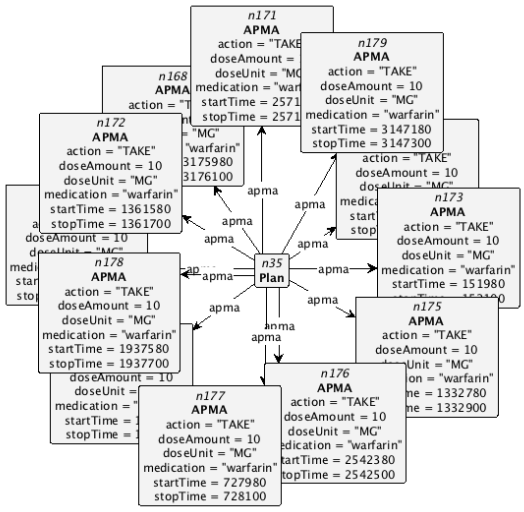
(a) After first three steps of time unrolling



(b) After application of recursive rule once



(c) After application of recursive rule twice



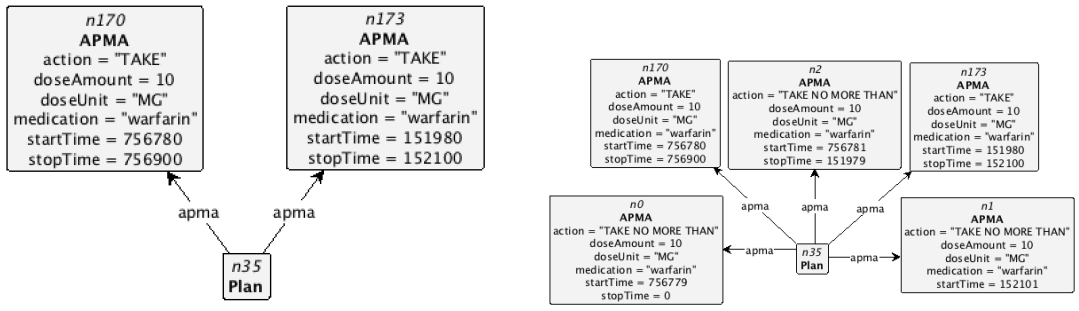
(d) After application of base case

Figure 4.15: Example of applying the rule that expand timing components.

4.4 Summary

This chapter presented a novel model of representing prescriptions as graphs. This model consists of the Rx Graph Model and the coreRx Graph Mode. The Rx Graph Model uses a set of nodes to describe the timing and dosing of the prescription. This graph is transformed by the presented graph transformation system into the coreRx Graph Model. The coreRx Graph Model specifies the entire lifetime of the prescription via a series of Atomic Prescribed Medication Actions (APMAs). Each APMA consists of: an action, a medication, an interval in time, and a dose.

The transformation of the Rx Graph Model to the coreRx Graph Model was broken down into four main steps, these are: static semantic checking, time unrolling, temporal unsharpening, and plan completion. Of these steps, the time unrolling is the most complex, using a recursive algorithm to compute the timing of each APMA. A



(a) Before application plan completion (b) After application of plan completion

Figure 4.16: Graphs before and after plan completion.

discussion of the strengths and weaknesses of the graph models and the transformation system is given in Chapter 6.

Chapter 5

Model of Medication Adherence

The previous chapter described two graph models and how one (the coreRx Graph Model) could be produced via graph transformation from the other (Rx Graph Model). This is one of the major components of the system that define the semantics of the DSL presented in Chapter 3. However, without defining a link to the "real-world" or reality of prescription, the semantics of the graph models (and by extension the DSL) would not be fully defined.

To complete the semantic definition of the graph models and DSL, a second graph transformation (and associated graph models) are required. These use the concept of a Triple Graph Grammar [42] to map between the coreRx Graph Model (referred to as the *plan graph*) and a graph describing how the prescription was actually administered (called the *administration graph*). The result of this mapping is a count of the number of APMAs (from the plan graph) that were actually adhered to by the patient. As well as providing a semantic definition for the DSL that is grounded in reality, this approach also provides a precise definition of medication adherence and makes it easy to measure.

In order to model the administration of a medication as a graph there must be a way to observe the medication administration events undertaken by the patient. This can be accomplished in many different ways. The Internet of Things paradigm envisions a vast number of sensors and actuators that are part of a person's environment. Such devices are already in prototype stages, and some show promise for being available to consumers within the next 5 years [46, 43, 48]. These devices would certainly be capable for creating a "trace" of the execution of a prescription, which could be used to generate the administration graph required for the approach described directly above. Such devices are subsequently referred to as "administration

sensors".

5.1 Graph Model for Adherence

As mentioned above, the system for mapping a prescription plan to an administration requires a new graph model. The graph model in question is Triple Graph Model that uses a set of mappings node to connect two distinct graph models. The GROOVE type graph of this model is presented in Figure 5.1.

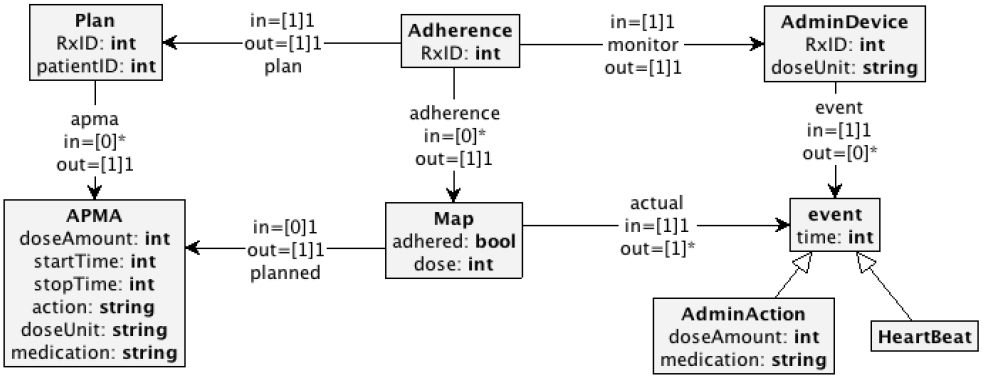


Figure 5.1: GROOVE type graph of the adherence triple graph model.

In left of Figure 5.1 is the coreRx Graph Model described in Chapter 4. The middle two nodes (**Adherence** and **Map**) describe the mapping function of the triple graph grammar. The **Adherence** node provides a node for the mapping nodes to connect to. The **Map** node type associates an **APMA** (from the coreRx Graph Model) to exactly one **event** node type in the administration, and is marked with an *adhered* flag to indicate whether the mapping between the **APMA** and **event** is considered medication adherence or non-adherence.

The **event** on the right of Figure 5.1 has two valid sub-types, either a **HeartBeat** or a **AdminAction**. The **AdminAction** describes a dose of medication taken at the indicated time, which is similar to the notion of an APMA but for administration rather than a plan. The **HeartBeat** event is meant to be emitted by the administration sensor at a regular interval (preferably one for every APMA), and has two purposes: i) to indicate that the administration sensor is still active; and ii) to provide measurements for adherence where no other administration event was recorded by the sensor. All **events** are associated with an **AdminDevice** that is

observing the administration of the medication. The AdminDevice node represents the administration sensor(s) described above.

5.2 Computing Adherence

Using the graph model(s) described above, a series of mapping rules can be created. Six mapping rules were created, these rules related only to the actions of "TAKE" and "TAKE NO MORE THAN" (abbreviated to "DONT" in the images). These rules are shown collectively in Figure 5.2.

Figure 5.2a describes a situation where an event is recorded (for the first time) within the interval specified by the **APMA**. This results in a new mapping action being created and the **APMA** and **AdminEvent** being connected. It may be the case that a patient takes a portion of their dose of medication at the beginning of the interval, and then the remainder at the end of the interval; such a case is covered by the rule in Figure 5.2c which checks that a new **AdminAction** does not exceed the allowed dose for that time interval. Should the dose exceed the allowed amount the **Map** node is marked as as non-adherent.

In the event that a **AdminAction** does not occur when there is a planned **APMA**, the rule in Figure 5.2e applies. This rule uses the **HeartBeat** concept to indicate that the device did not detect a administration event. This is the one of the possible ways in which non-adherence can occur. Conversely, the rule in Figure 5.2f applies in cases where a **HeartBeat** occurs that corresponds with a "DONT" **APMA**. Such a mapping constitutes adherence, since the patient was not observed to have taken their medication, nor should they have.

The rule in Figure 5.2b applies when an **AdminAction** occurs but the corresponding **APMA** describes a negative or "DONT" take action, which also constitutes non-adherence. This rule is complemented by 5.2d, which provides a mapping to a specific **APMA** for several possible non-adherent **AdminActions**. These two rules check the *doseAmount* field in the **AdminAction**, this is required because of the action "TAKE NO MORE THAN". It may be permissible for a patient consume some small amount of a medication over some time period, provided they do not exceed a specific amount.

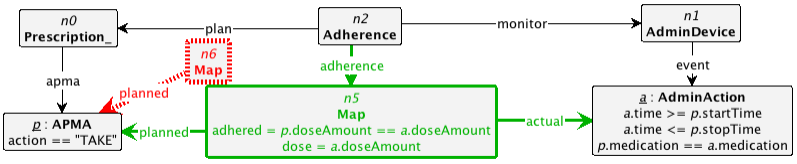
The result of using this transformation to map between the plan and administration graph is a set of **Map** nodes that connect **events** from the administration sensor to **APMAs** from the prescription plan. These **Map** nodes are annotated with

adherence flags. From this, an adherence measure or score can be calculated for the prescription in question. A simple approach for calculating the adherence is to find the ratio of "true" or positive adherence mappings to the total. This can be expressed as:

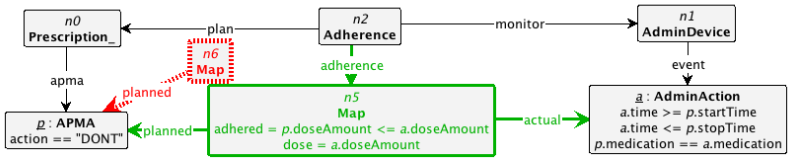
$$A = \frac{Count(\text{positive adherence events})}{Count(\text{adherence events})}$$

5.3 Summary

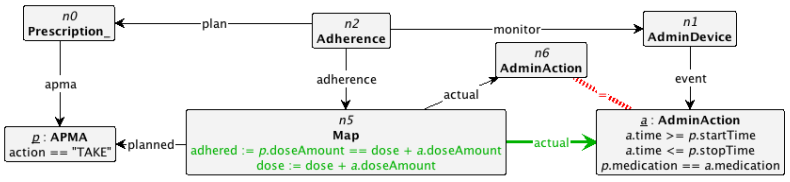
This chapter introduced another graph model and transformation system that maps the concept of a prescription plan to the actual administration of the prescription. The result of this is three-fold: i) the semantics of the Rx Graph Model, coreRx Graph Model, and domain specific language for prescribing are all formally specified with respect to reality; ii) the meanings of adherence and non-adherence are precisely defined; and iii) a method of computing the degree of adherence to a prescription is defined. This work represents a novel approach to describing and measuring medication adherence. Further discussion of the concepts from this chapter are provided in Chapter 6.



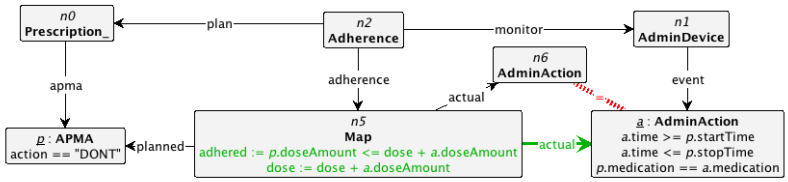
(a)



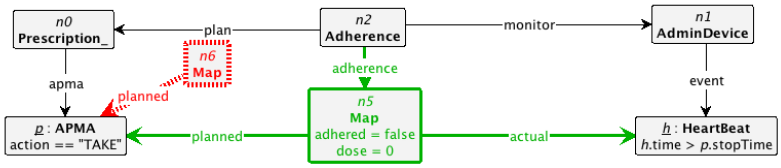
(b)



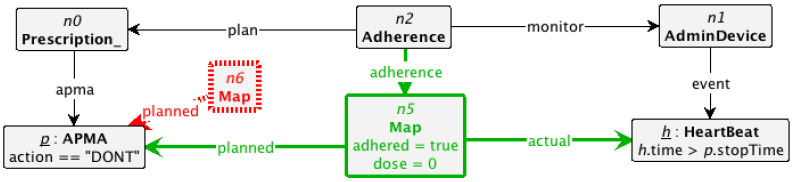
(c)



(d)



(e)



(f)

Figure 5.2: GROOVE triple graph grammar rules describing the mapping between the plan and administration graphs.

Chapter 6

Discussion

Thus far, this work has presented a domain specific language (DSL) for authoring medication prescriptions; two graph models (Rx Graph Model and coreRx Graph Model) for expressing prescriptions; a graph transformation system for converting the Rx Graph Model and coreRx Graph Model; and a triple graph system for recoiling a planned prescription with the trace of the reality. Combining the graph transformation systems with the DSL provides a means of formally specifying the semantics of the DSL by linking the reality trace of the prescription with the plan. To my knowledge, a DSL for authoring prescriptions backed by a formal graph model is unique. This chapter provides a discussion of each of the pieces of work presented in the previous chapters.

6.1 Domain Specific Language

As previously mentioned, one of the goals of developing a DSL for prescribing was to allow for prescriptions to be described in a form that is both expressive enough to support the range of concepts in present in prescribing, and structured enough to allow for a computer program to process it reliably. The overall structure of a prescription created using DSLs was created to represent a natural language expression of a prescription. Specifically, the aforementioned structure is:

<action> <medication> <dose> <timing> <repeat>

Such a structure naturally represents a natural language prescription such as “take aspirin 81 mg once daily”. Admittedly, such a prescription is quite simple and does

not represent the full gamut of concepts found in primary care prescriptions. This becomes clear when examining a prescription such as "take warfarin TITRATE down FROM 10 mg TO 0 mg BY 1 mg once daily FOR 10 days" where the line between the dose and timing concepts is not fully clear. Upon reflection, the cause of this is the fact that the concept of titrating doses combines the the concepts of dose and timing.

Examining another example reveals another weakness in the current language design, namely that when describing specific timing and doses, there can be uncertainty about how the timings and doses relate. Consider the example: "take aspirin (81 mg, 162 mg) twice daily (8, 20) FOR 10 days", it may not be immediately clear whether the "81 mg" dose matches the "8" or "20", furthermore it is not clear that the numbers that indicate specific timings represent hours during the day.

Though significant effort over the course of this project was dedicated to making the DSL usable for clinical users, the only feedback on the language structure itself was from a domain expert who was a primary care physician. While this feedback was invaluable in directing the effort and defining the semantics of the DSL, it represents a single user's perspective and is far from a formal empirical evaluation of the DSL. Further examination of the DSL and detailed evaluation of its use by a range of clinical users is required before the current prototype can confidently be used in practice.

Furthermore, the set of prescriptions that can be authored in the prescribing DSL is a *subset* of the prescriptions that can be described by the Rx Graph Model. For example, the Rx Graph Model can specify nested time frames (a specific hour inside of a particular day inside of a particular week); such a prescription can not be described by the current implementation of the DSL. The implications of this particular case in clinical use are not immediately clear, however it is likely that most prescriptions in primary care do not require such a detailed level of specification, and addressing these timing concerns in the current language would affect only a small percentage of prescriptions. Of course, he goal of the DSL is to allow for structured input of these more complex prescriptions. Therefore the DSL should be re-structured to support the entire set of prescriptions that can be specified by the Rx Graph Model, and adjusted as required to support new concepts as they are introduced into the Rx Graph Model.

The final critique of the DSL is the fact that it has been targeted for primary care prescribing, and within that, a subset of primary care prescriptions. Within primary care, there is a wide range of prescribing concepts, and some of which have not yet

been captured by the DSL. For example, an insulin prescription, which typically has complex administration instructions with many different conditions, would not readily fit within the current DSL. Furthermore, the domain of primary care is one of many different domains in health care where e-prescribing occurs. Future changes to the DSL (and associated graph models) could focus on applications in secondary and tertiary care settings.

6.2 Graph Models and Transformation

The two graph models presented for describing prescriptions and the transformation system presented in this work provide a means of grounding the semantics of the DSL in the real world concepts of prescribing. One of the important steps of the transformation between these two graph models is the ability to apply static semantic checks to the input Rx Graph Model prior to transformation to the coreRx Graph Model. While current checks are focused on simple syntactic and semantic evaluation of the input, there is significant potential to evaluate a wider range of prescribing concepts using both static and dynamic checking. Such processes could lead to a system that prevents erroneous prescriptions from being authored, and would ultimately ensure greater safety in the medication management process.

One of the advantages of using a graph model and transformation system backed by a rigorous mathematical constructs such as graph theory and graph transformation is the ability to conduct proofs of correctness on the system in question. Chapter 2 describes several properties of graph transformation systems (termination, confluence, convergence, and correctness) that are of interest in this work. Unfortunately, conducting formal proofs of such concepts is very challenging in complex graph transformation systems and was not completed during this project. As a result, these properties were not *proven* in the general case for the entire graph transformation system. The implication of this are that one cannot say with mathematical certainty that any valid input will produce a correct output; in fact there could be many different configurations of input graphs that conform to the Rx Graph Model constraints, but would violate the desired properties of a transformation system listed previously. Therefore, one cannot claim that the current graph models and transformation system provide a *total* formal semantic definition of the prescribing DSL. Future work is required to exploit the full rigor of the mathematical grounding of graph theory and graph transformation.

Though total definition of semantics is extremely difficult for this particular problem, it is still possible to show that the system is a *partial* formal semantic definition. One can use the GROOVE tool to explore the state space of the graph transformation system and validate that desirable properties for the system hold for a specific input. GROOVE's state space exploration tools will apply all rules in all possible orders and report the resulting set of "final" states of the graph model. If exactly one final state is found, then the transformation was convergent on that state for that particular input. By testing a variety of inputs, it is possible make a more confident statement about the properties of the system, however this does not constitute a proof. Conceptually, this process is similar to any other software testing methodology, in the sense that it does not guarantee software correctness, but rather just validates the software for a particular set of inputs and/or conditions.

The current implementation of the graph transformation system has been tested in the manner described above with a series of test cases, some of which are given as examples in Chapters 3 and 4. Future work on this project may focus on proving the properties of transformation system, which would result in a total formal semantic definition of the DSL.

The current graph models (Rx Graph Model and coreRx Graph Models), described in Chapter 4, have several weaknesses or areas for improvement/extension. The first is the notion of a "condition action". Many prescriptions may specify an action as a function of some external factor, this is common in prescriptions for insulin where the dose administered changes depending the most recent blood sugar level or intake of food. Such a concept could be added to both graph models, and would likely be an extension of the **Action** component. The concept of a conditional action would then need to be designed into the DSL.

Another area for improvement of the graph models is the concept of "concurrent" prescribed actions, more specifically, prescribed actions that are meant to be carried out at the same time. The current Rx Graph Model assumes that actions occur independently in time. This assumption was made to simplify the problem; such simplifications and assumptions are the reality of modeling any sufficiently complicated real-world process. The Rx Graph Model and transformation system would require the most change to support concurrent actions, since there is currently no restriction on the coreRx Graph Model for overlapping time intervals. Thus changing the coreR Graph Model would not require significant effort. In a sense, the current Rx Graph Model supports concurrency simply by demanding that one graph match with one

prescription, however this then requires each prescription have its own Rx Graph Model. It may be more efficient to allow prescriptions that share the same timings to be specified within the same graph model. This concept could be the subject of further work.

With respect to the medication component of the graph models, not all concepts have been fully specified. Thus far, the assumption made was that the medication had the form, route, brand name, generic name, and substance(s) all combined into one "super" medication concept. In reality, these items are important parts of the prescription that must be specified by the clinician. These concepts were originally omitted from the current graph models as they are simple enough to be added later as attributes to the medication component. However, it might be desirable to consider these concepts as variables that change with dose or time rather than constants. For example, one may want to specify different routes for different periods of time (intravenous for 1 week, oral for 7 weeks). Specifying more detailed medications would likely require change to the DSL, Rx Graph Model, and transformation system, but not to the coreRx Graph Model.

Finally, it is not clear how the current graph models support different roles in the medication management process. Physicians are the primary prescribers of medication, however they are not the only clinical actors that can affect change on the medication management process. In many situations, pharmacists and nurses are required to refine or modify prescriptions during the act of dispensing the medications. The current model does not support the notion of dispensing of a prescription. To support these concepts the current models and transformations need not change directly; instead they must be extended with other transformations that describe the acts of modifying and dispensing prescriptions.

The current graph models and transformation system support the concepts of timing and dosing quite well given the fact that the system is a prototype meant to operate within a restricted scope in primary care. Indeed, the creation of model for timing and dosing was perceived as the largest challenge in this project. Further evaluation is required to ensure that the assumptions made within the models are accurate representations of the real-world. This evaluation would likely be in the form of discussion with domain experts and examination of a large number of a prescriptions of varying complexity from a real-world primary care practice.

6.3 Model of Adherence

The graph model(s) and triple graph grammar system for reconciling observed medication administration actions with the planned prescription actions represents one of the novel aspects of this work. Associating the plan and administration graphs provides a means of describing the semantics of the Rx Graph Model, coreRx Graph Model, and DSL, and gives a precise definition of what adherence means. Such a definition is extremely valuable as it can be used to inform work on medication adherence in many fields. For example, the definition for medication adherence presented by the World Health Organization is: *"the extent to which a persons behaviour taking medication, following a diet, and/or executing lifestyle changes, corresponds with agreed recommendations from a health care provider."* [31]. Such a definition is extremely imprecise, and, amongst other problems, makes it difficult to define what is required for solving this problem of "striking magnitude".

Though this model provides *a* definition for adherence, it does not necessarily provide the *best* definition. The current model and transformation rule set are quite rigid in what constitutes adherence. If a dose is taken within the time interval defined by the APMA then it is valid, if it was even one second later it would be invalid. Drawing such a stark dividing line between these two points in time seems naive. Other approaches that use a more relaxed (in the context of timing) version of adherence/non-adherence must be considered. One possible approach is to use the concept of Fuzzy Logic or Fuzzy Sets, in which values are neither "in" nor "out" of a set, but fall onto some distribution. Such an idea is conceptually possible to implement. A similar problem exists for the concept of doses of medication, again the solution may lie along the lines of Fuzzy Logic.

Another concern with the current model of adherence is how it adapts to change in the planned regime. This is best illustrated by an example:

> *Suppose a patient is asked to take their medication once every 24 hours. The patient takes their medication in the morning as prescribed (full adherent) for one week, then misses their dose on Friday morning (because they are tired from staying up all night writing their thesis); however the patient takes their medication as soon as they remember even though they were late by about half a day. They continue to take their medication as prescribed (every 24 hours) but are now taking their medication in the afternoon rather than the morning.*

In the above example, the patient would be considered almost fully adherent, the exception being their late dose on Friday. However, under the current model, all of their doses after the first dose they missed would be counted as non-adherent. This is significant weakness in the model. One could solve this problem by simply extending the time intervals to be so broad that they allow for significant "wiggle" room, however this approach would degrade the precision of the measurements, and if taken too far would not be able to provide *any* meaningful measurement of adherence. Another approach to solving this would be to consider a prescription as a dynamic plan that *changes* given different varying administration actions.

The concept of a dynamic plan, or feedback loop, for a prescription plan requires that the prescriber also provide a series of instructions for how to handle missed or incorrect doses. These are termed "re-adherence" instructions, as they would ideally describe how one could return to adherence after deviating from the prescription plan. For some prescriptions, such as vitamins, these actions may be trivial. For example, if one misses their daily dose of Vitamin C one or twice, the effect on their health is not likely to be catastrophic. It would be sufficient to count the missed dose and continue to the next, this would constitute a "skip" action. Other medications might be more serious, the concentration of warfarin (a "blood thinner") in the blood for example, must be consistently high enough have an effect, however too much can cause significant harm. A prescription for such a medication would need to be accompanied by a more comprehensive set of re-adherent instructions.

6.4 Applications

The application of the system presented in this thesis is two-fold: i) the domain specific language, once further developed and evaluated, can be used to support clinicians in their prescribing workflow; and ii) the model of adherence can be used as the core component in an information system that tracks and evaluates adherence. Furthermore, components each of these applications can be combined to create systems that have other value propositions. For example, the DSL and graph models could be used as input to a decision support system that reasons not about adherence, but about the timing of medications and how they can distributed to reduce drug-drug interactions or to fit the patient's lifestyle.

A system that uses all of the functionality described could act as an end-to-end medication management system, provided a reliable method of sensing medication

administration events exists. Such as system could link a clinician's information system with a pharmacy, and then with a series of environmental sensors. The administration events observed by the sensors could then be used to provide feedback to patients and clinicians about the adherence of the patient to the prescription, which can ultimately lead to care that better supports the patient. This system could be further augmented by the feedback loop described in the preceding section, which would allow patients to recover from slipped or missed doses.

Chapter 7

Related Work

This chapter is intended to provide an informal review of current work in areas of study as they are related to the topic of this thesis. It is not meant to be an exhaustive review, but rather attempts to capture the main themes in these areas. The resources for this review were generated by searching appropriate databases (PubMed, Engineering Village), and through reference mining, no formal literature review procedure was used.

7.1 Similar Work

To my knowledge, work by Yeh *et al.* is the only other work that has attempted to create both a domain specific language (DSL) for prescribing medications and an accompanying data model [52]. Yeh *et al.*'s work focused on developing a DSL for authoring prescriptions for the purpose of identifying conflicts between medications called APAMAT (A Prescription Algebra for Medication Authoring Tool). It was designed to evaluate sets of prescriptions for potentially harmful drug-drug interactions and to generate prescription schedules. APAMAT allows for specification of a variety of medication concepts, with an emphasis on specifying max and min values of doses and time intervals. However, their proposed language does not support more advanced concepts such as tapering of medication doses. Furthermore, APAMAT uses a notation that is too verbose for use in by a clinical user. This notation consists of numerous structures and characters that would be familiar to a programmer, however are not likely to be adopted by a clinical user. Finally, the underlying semantics of APAMAT are not specified (to my knowledge) by any formal model or link to reality,

this limits the amount of reasoning that can be completed by the tool. Despite this, it was an excellent resource and was a starting point for much of the work in this project.

At this point, the only person to precisely define a definition for medication adherence is Varsheny [48]. His model, similar to this work, was geared towards adherence in primary care. Varshney defines adherence using a probabilistic model to express the probability of events occurring given a number of factors including reminders, functionality of administration sensors and actuators, and the patient's history. This is distinctly different from the model presented in this work and considers many factors that our model cannot such as how infrastructure (devices, networks etc.) failure rates can affect adherence. However, the concepts in Varshney's model are not expressed in a computational model, making it difficult to use in practice. Nevertheless, this model provides a number of ways in which our model(s) can be extended to encompass more of the factors affecting medication adherence.

7.2 e-Prescribing

In general, IT tools for e-prescribing have been received with a positive but weary attitude. The 1999 report "To Err is Human" by the Institute of Medicine spurred an effort to create e-prescribing systems [1]. It is evident that the technology has the potential to benefit the prescribing process, not only in terms of increasing patient safety, but also with cost and time savings. Goundrey and Smith's book "Principles of Electronic Prescribing" outlines the current state of e-prescribing, their chapter "Philosophical and Social Framework of Electronic Medicines Management" is an excellent summary of the value e-prescribing can bring to the clinical setting, including advanced decision support, increased accuracy, communication, and standardization [17].

Adoption of e-prescribing systems is slowly increasing due to a variety of factors, for example incentives offered by professional colleges and governments [7]. Fischer *et al.* studied the adoption of e-prescribing systems over the course of 12 months in community based clinical practices. They found that the rate of use increased as physicians became more comfortable with the technology, however, the rate of adoption was below 30%. This low adoption rate appears to be tied to the effort required to learn a new skill in an already busy environment, especially for clinicians who have been using paper for the majority of their career [11]. Another study by

Randhawa *et al.* examined the use of e-prescribing systems in a group of primary care physicians and found that they used most of the e-prescribing tools available to them, however they were far from obtaining ideal use [37].

As with any technology, new hazards and risks are uncovered as it is put to use in a real world setting. HIT systems have not yet been elevated to the status of safety-critical, however the evidence to consider it as a safety critical system is rapidly growing. Wears and Leveson discuss the evolution of software in health care and its impacts, and describe how a change in the safety status of health information software can be brought about by considering these systems as complex sociotechnical systems that have safety built in from the start [51].

Wears and Leveson's sentiment is echoed by others who feel that the hazards of this new technology need to be understood by all parties involved. Borycki explores the concept of technology induced errors and describes a framework that supports recovery and improvement based on these errors [7]. Her concerns are supported by findings of Campbell *et al.*, Ash *et al.*, Slight *et al.*, and Kaushal *et al.* all of which present errors or failures due to e-prescribing systems and call for changes in the way these systems are implemented [8, 3, 44, 24]. Most shockingly, increased mortality rates were observed in a hospital after a new e-prescribing system was implemented [20]. Odukoya and Chui also characterized hazards in e-prescribing systems, but from the point of view of pharmacists, and found that they fell into three broad categories 1) increased cognitive burden, 2) workflow disruptions, and 3) communication problems leading to errors [30]. Koppel *et al.* have documented a variety of errors that stem from e-prescribing systems, the errors they report are related to the interaction of clinicians with the e-prescribing system, with several relating directly to the presentation of information.

Most of the work discussed above attributes adverse medication events and prescribing errors (at least partially) to flaws in user interfaces. A detailed analysis of a single adverse medication event by Horksky *et al.* also found that user interface design had significant impact safety of the system [22]. Clearly, user interface design plays a significant factor in success or failure of e-prescribing systems. Horsky acknowledges this and has described the optimal features for clinical decision support system notifications. Horsky emphasizes understanding the cognitive loading of users when using health information systems, and that interface should support their mental processes.

As would be expected, changes to user interface have been found to affect changes in the quality of data that is entered during their use. Specifically, Turchin *et al.*

found that changing the user interface changed the number of internal discrepancies of prescriptions that were entered [47]. Their results were similar to the findings of Palchuk *et al.* which indicate that the use of free text fields to input special instructions contributes to the number of internal discrepancies in a prescription [33]. Both of these results coincide with the main purpose of this thesis, they indicate the need for a more expressive interface feature that can capture the complexity of prescribing.

The ideal approach to prescribing would be a tool that interprets free text using Natural Language Processing to extract important information. This is a very challenging task as the number of possible inputs is quite large. There has been significant effort spent to develop tools that are capable of doing this, Friedman summarizes the key concepts of natural language processing in medicine in their book [12]. The field of natural language processing is vast and was not explored in any more detail, it was mentioned here for completeness.

In summary, the use of e-prescribing systems is increasing. It is evident that these systems are capable of reducing classes of errors related to traditional paper prescribing, however new classes of errors and failures are emerging as these systems become increasingly relied upon. While these systems are not yet considered safety critical, they certainly pose a risk to those who are impacted by their use. User interface design has been identified as one of the main sources of errors in e-prescribing, current research calls for further evaluation and revision of existing user interfaces. One of the user interface elements that has been a cause for concern is free text entry fields, which allow for non-structured information to be entered into the system. There is not yet a proven solution for managing data entry from free text fields, however natural language processing and domain specific languages have been proposed as method for structuring free text information.

7.3 Graph Transformation

Graph rewriting or graph transformation as a formalism was first introduced by Pfaltz and Rosenfeld in 1969 to describe changes to graphs instead of transformation to Strings [36]. Graph transformation has been developed over a number of years with major contributions from Pratt and Ehrig *et al.* in the 1970s [45]. In a paper giving an overview of the field, Toffetti describes several stages that a formal method/approach (such as graph transformation) must go through before it is widely practiced [45].

Graph transformation, according to Toffetti, is in a stage where it has matured in the academic community and is waiting for the right "killer application" before it will be accepted more broadly into industrial practices [45].

Graph transformations have value as a formalism for describing both static and dynamic systems. They are based on graphs, they are a natural model for representing many different real-world phenomena [5]. Applications generally fall into four categories: i) modeling software systems, ii) defining semantics of domain specific languages, iii) mapping between different notations and languages, and iv) proving and checking properties of systems [45]. Specific applications include role based access control models [25], simulating chaotic phenomena [13], and modeling autonomous rail systems [21] to name a few. A number of applications are presented in the 2nd volume of the *Handbook of Graph Grammars and Computing by Graph Transformation* [9].

One of the major barriers to the adoption of graph transformation identified by Blostein *et al.* was the availability of tools to support complex real-world problems encountered by industry [6, 45]. Since then, a number of tools have been developed that support various aspects of graph transformation. Some tools, such as Fujaba, have become quite mature and have been used on large joint industry-academic projects [45], others such as GROOVE [40] have been developed with a focus on simulation. In a recent paper about GROOVE, Ghamarian *et al.* provide a comparison of 10 different graph transformation tools [13].

Chapter 8

Conclusion

This thesis has presented a domain specific language for authoring prescriptions. The semantics of the language were formally defined by using a series of graph models and transformation systems. The three graph models developed where the Rx Graph Model, the coreRx Graph Model, and an Administration Graph Model. These graph formalisms link the concepts in the domain specific language to reality. Grounding the models in reality also provided a precise definition of medication adherence. Using this formal definition of medication adherence, the adherence of a patient to a prescription can be computed.

The applications of this work are quite broad and range from improving clinical medication management and clinical decision support, to augmenting existing medication management devices with more advanced features that support dynamic medication regimes. There are many avenues for future work that could be pursued, some include: evaluating and revising the domain specific language; integration of the revised domain specific language with an active clinical information system; further refinement of the graph models and transformation systems to support more prescribing concepts; and integration of the current medication adherence features with existing medication administration sensing devices.

It has been my pleasure to work on this project for the past 8 months. I have learned an incredible amount about a plethora of topics. This would not have been possible without guidance and assistance from both supervisors, Dr. Morgan Price and Dr. Jens Weber. Dr. Price has provided guidance for management of this project (and other related projects), his expertise in both informatics and clinical medicine helped me understand (a small portion) of the complex topic that is prescribing medications.

While I was able to develop many of the concepts and technical prototypes during this project, Dr. Weber developed the triple graph grammar model described in Chapter 5. He also assisted with the other concepts by providing detailed review and revisions of the graph models. Without his guidance, this project would not have taken shape it did. He introduced the concept of graph transformation (amongst other formalisms) to the project, and in doing so taught me about a mathematical formalism that has changed the way I think about Software Engineering.

Appendix A

DSL Grammar

This appendix contains the ANTLR4 grammar for the prescribing domain specific language. ANTLR4 grammars are similar to the BNF of the grammar, however they allow for extra annotations and use a slightly different syntax.

```
grammar Prescription;

// START:tokens
INT      :   '0'..'9'+ ;

NUMBER   :   ('zero' | 'one' | 'two' |'three'|'four'|'five'|
    'six'|'seven'|'eight'|'nine'|'ten')+;

TIMEUNIT:   ('hour'| 'day'| 'week'|'month'|'year')+;

TIMEUNIT_PLURAL : TIMEUNIT's'+;

UNIT     :   ('mg' | 'g' | 'kg' | 'mcg' | 'ng');

INTERVAL_FREQ : ('once'|'twice'|'thrice');
INTERVAL_MODIFIER : ('per' |'times'| 'times per' |'x');
INTERVAL_LENGTH : ('daily'|'weekly'|'monthly'|'yearly'|'annually');

STRING   :   ('a'..'z'|'A'..'Z'|'-')+ ;
ID       :   ('a'..'z'|'A'..'Z')+ ;



NEWLINE  :   '\r'? '\n' ;
WS       :   (' '|'\t'|'\n'|'\r')+ {skip();} ;

script : expr+;

expr : expr 'THEN' expr | atom | expr NEWLINE | NEWLINE;

repeat: 'REPEAT' repeatValue | repeatValue 'REPEATS';

repeatValue: INT;

atom : action medication dose timing repeat?;

action : STRING;

medication : (STRING)+;

dose: titratingDose | fixedDose;
```

```
fixedDose: specificDose | doseAtom;

doseAtom: doseAmount doseUnit;

doseAmount: INT;

doseUnit:  UNIT;

specificDose: '('(doseAtom',')*? doseAtom')';

titratingDose:
    'TITRATE' titratingDirection titratingStart titratingStop titratingChange titratingInterval
;

titratingDirection: 'up'|'down';

titratingStop: 'TO' doseAtom;

titratingStart: 'FROM' doseAtom;

titratingChange: 'BY' doseAtom;

titratingInterval: 'per' durationAmount? durationUnit;

timing :
    interval specificTiming 'FOR'  duration
    | interval specificTiming | interval 'FOR' duration | interval;

instant :  INT;

specificTiming : '('(instant',')*? instant')'; //(8, 12, 14)

interval:
    frequency intervalLength
    | frequency  INTERVAL_MODIFIER  intervalLength
    | frequency
;

frequency: (INTERVAL_FREQ|INT|NUMBER) ;

intervalLength:  (TIMEUNIT|INTERVAL_LENGTH|TIMEUNIT_PLURAL); //once daily, twice weekly etc...

duration: durationAmount durationUnit;

durationAmount: (NUMBER|INT);

durationUnit: (TIMEUNIT_PLURAL | TIMEUNIT);
```

Appendix B

Graph Transformation System

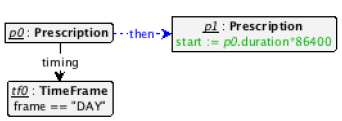
B.1 Control Program

```
while (runCondition){main();}

function main(){

        //Static Semanitc Validation
        checkConstraints();
        createPlan;

        //Time Unrolling
        expandThen();
        expandTitrate();
        expandRepeats();
        expandDurations();
        expandTimeFrames();

    //Plan Completion
    completePlan();
}

function completePlan(){
    alap plan_completion_1;
    while(apma_then_condition){
        try{plan_completion_2;}
        try{plan_completion_3;}
    }
}

function expandThen(){
        while(thenCondition){
                try{expandThen_day;}
                try{expandThen_week;}
                try{expandThen_month;}
        }
}

function checkConstraints(){
        if(check_frequency){
                try{
                        failureRule_frequencyMismatch;
                        failureExit;
                }
        }
        if(check_TimePointValues){
                try{
```

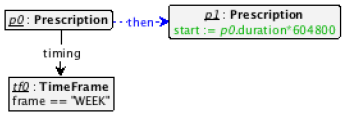
```
                        failureRule_duplicateTimePoint;
                        failureExit;
                }
        }
}

function expandTitrate(){
        while(titrateCondition){
                try{expandTitrate_day_1;}
        }
}

function expandDurations(){
        while(durationCondition){
                try{expandDuration_month_1;}
                try{expandDuration_week_1;}
                try{expandDuration_day_1;}
                try{expandDuration_hour_1;}
        }
}

function expandTimeFrames(){
        while(compoundTimeFrameCondition){
                expandCompoundTimeFrames();
        }

        while(multipleAtomicEventsCondition){
                try{copyAtomicEvents_1;}
                try{copyAtomicEvents_2;}
        }

        while(atomicTimeFrameCondition){
                if(titrateCondition2){

                        while(titrateCondition3){
                                try{orderPrescriptions_titrate_1;}
                        }

                        expandAtomicTimeFramesWithTitrate();
                }else{
                        expandAtomicTimeFrames();
                }

        }
}

function expandAtomicTimeFrames(){
        try{expandTimeFrame_Atomic_day_1;}
        try{expandTimeFrame_Atomic_hour_1;}
        try{expandTimeFrame_Atomic_week_1;}
        try{expandTimeFrame_Atomic_month_1;}
}

function expandAtomicTimeFramesWithTitrate(){
        try{expandTimeFrame_Atomic_day_titrate_1;}
        try{expandTimeFrame_Atomic_day_titrate_2;}
        try{cleanTitrate_1;}
}

function expandCompoundTimeFrames(){
        try{expandTimeFrame_Compound_month_1;}
        try{expandTimeFrame_Compound_week_1;}
        try{expandTimeFrame_Compound_day_1;}
}

function expandRepeats(){
        while(repeatCondition){
                try{expandRepeats_month;}
                try{expandRepeats_week;}
                try{expandRepeats_day;}
        }
```

```
}
```

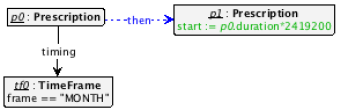
B.2 Rules



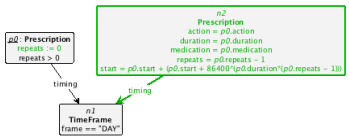
(a) Rule expand_then_day



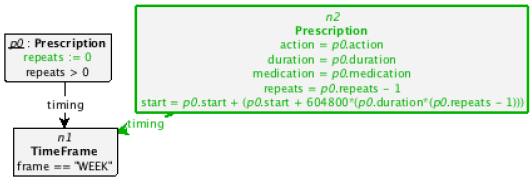
(b) Rule expand_then_week



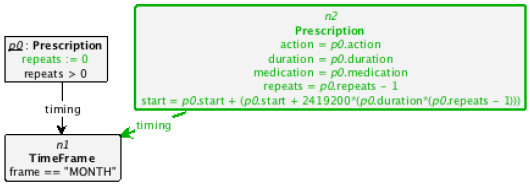
(c) Rule expand_then_month



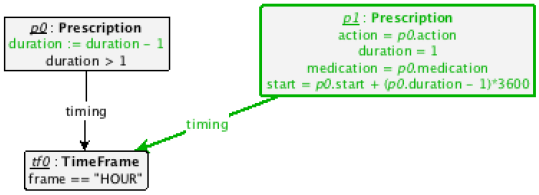
(d) Rule expand_repeats_day



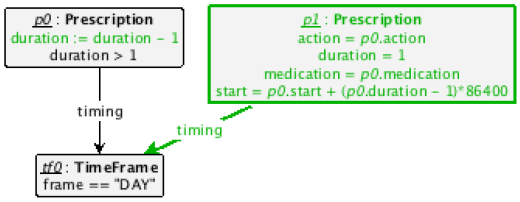
(e) Rule expand_repeats_week



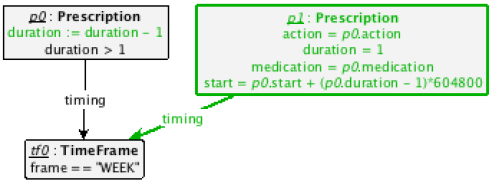
(f) Rule expand_repeats_month



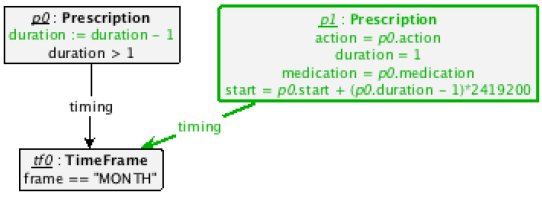
(g) Rule expand_duration_hour



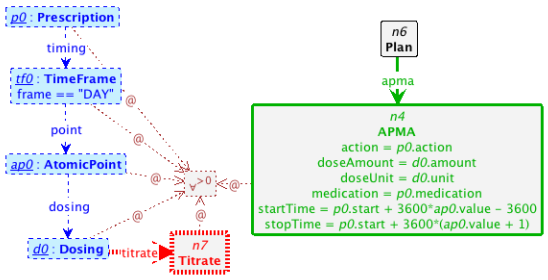
(h) Rule expand_duration_day



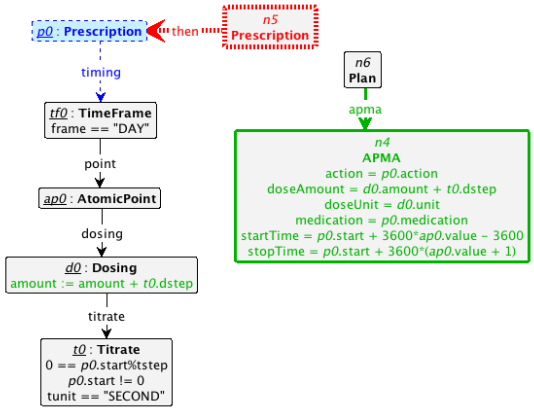
(i) Rule expand_duration_week



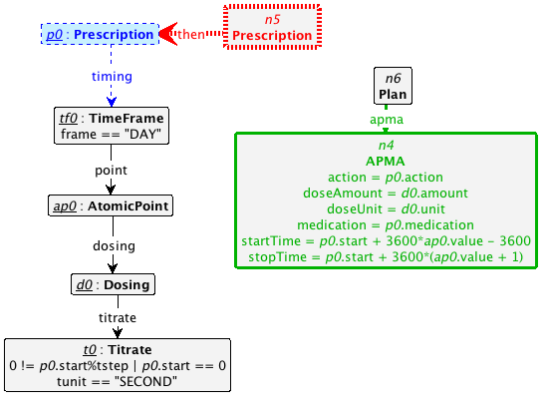
(j) Rule expand_duration_month



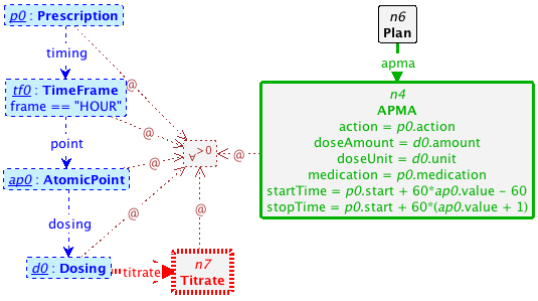
(a) Rule expand_time_frame_atomic_day



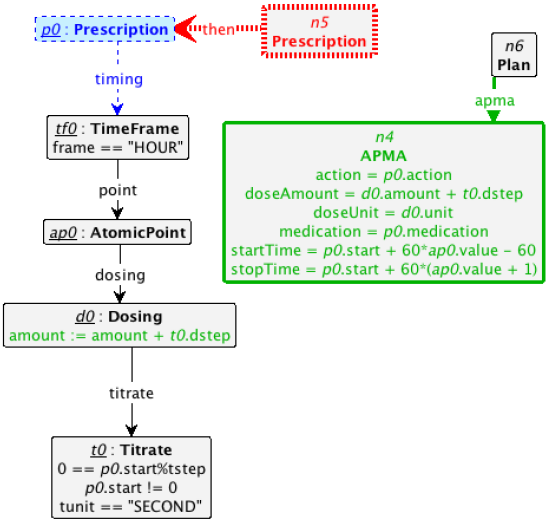
(b) Rule expand_time_frame_atomic_day_titrate_1



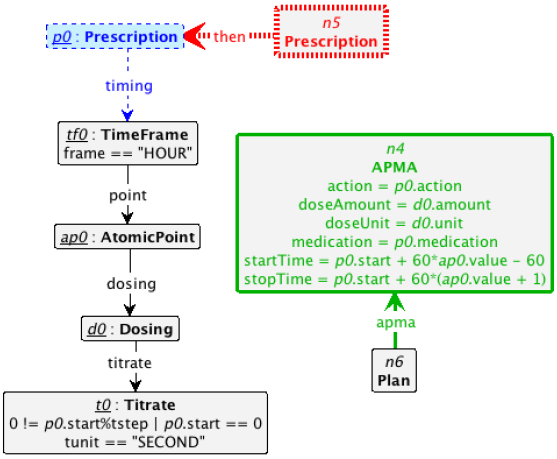
(c) Rule expand_time_frame_atomic_day_titrate_2



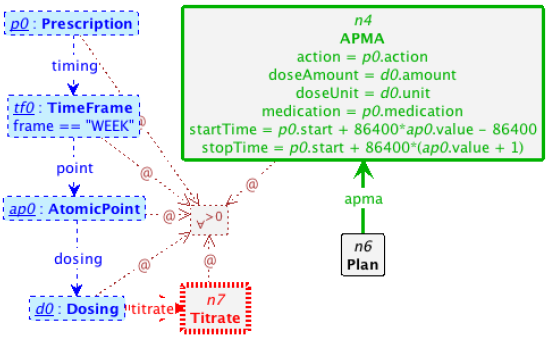
(d) Rule expand_time_frame_atomic_hour



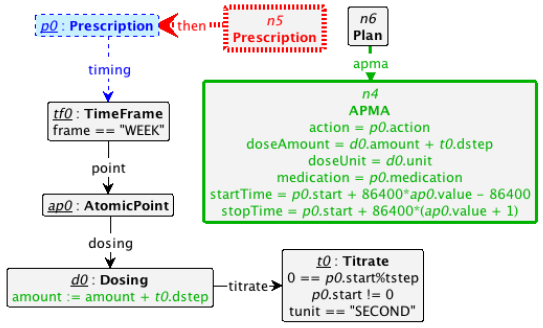
(e) Rule expand_time_frame_atomic_hour_titrate_1



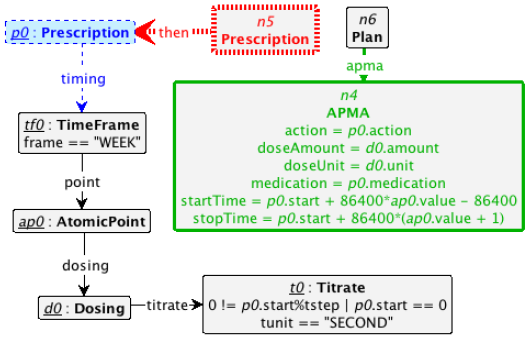
(f) Rule expand_time_frame_atomic_hour_titrate_2



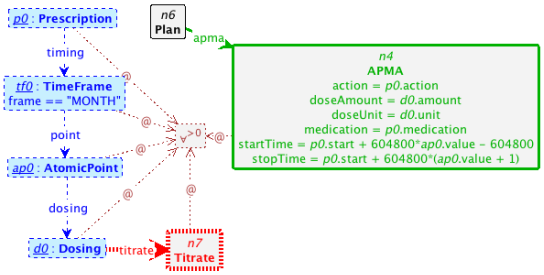
(a) Rule expand_time_frame_atomic_week



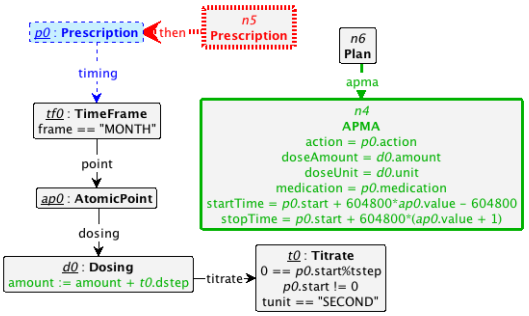
(b) Rule expand_time_frame_atomic_week_titrate_1



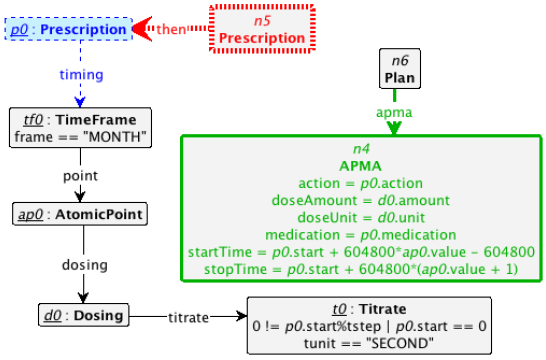
(c) Rule expand_time_frame_atomic_week_titrate_2



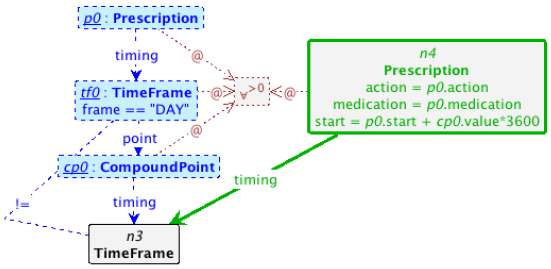
(d) Rule expand_time_frame_atomic_month



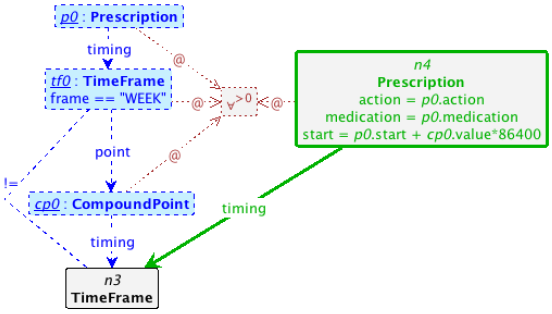
(e) Rule expand_time_frame_atomic_month_titrate_1



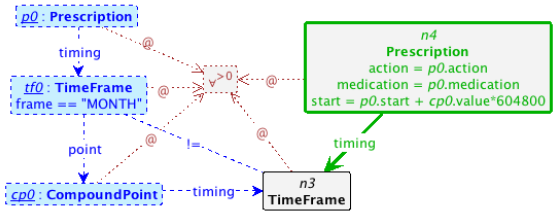
(f) Rule expand_time_frame_atomic_month_titrate_2



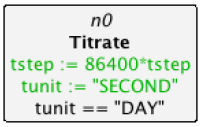
(a) Rule expand_time_frame_compound_day



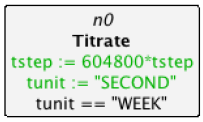
(b) Rule expand_time_frame_compound_week



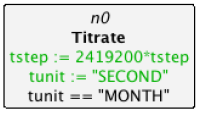
(c) Rule expand_time_frame_compound_month



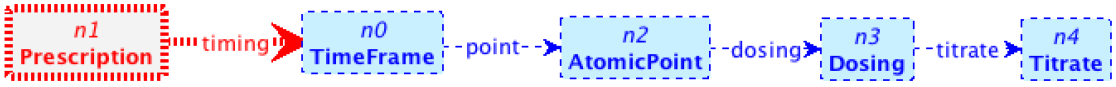
(d) Rule expand_titrate_day



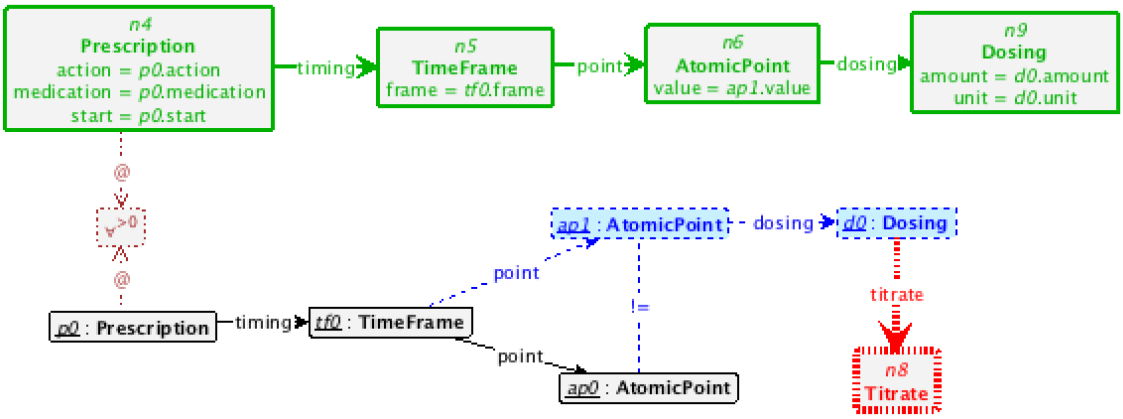
(e) Rule expand_titrate_week



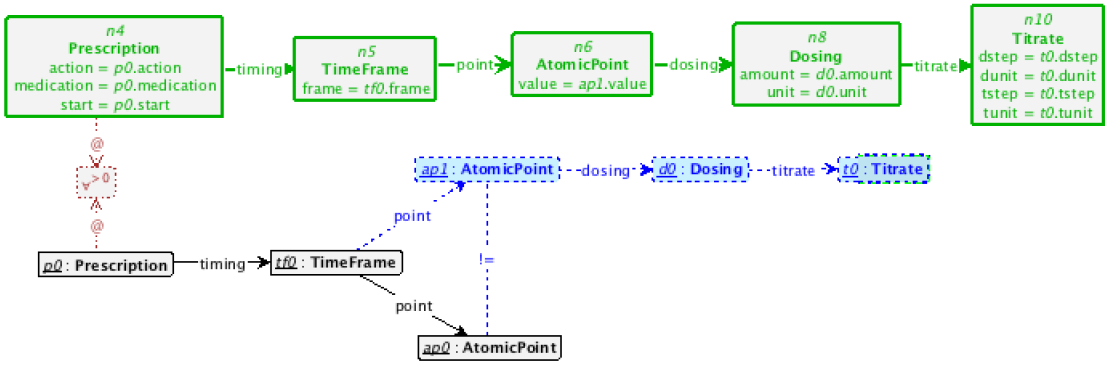
(f) Rule expand_titrate_month



(a) Rule clean_titrate

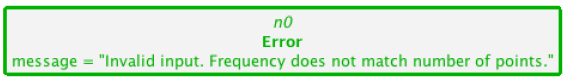


(b) Rule copy_atomic_events_1

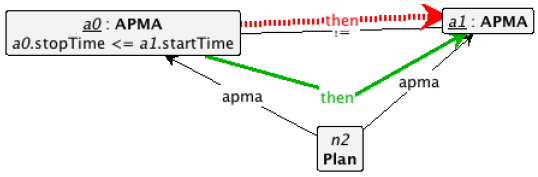


(c) Rule copy_atomic_events_2

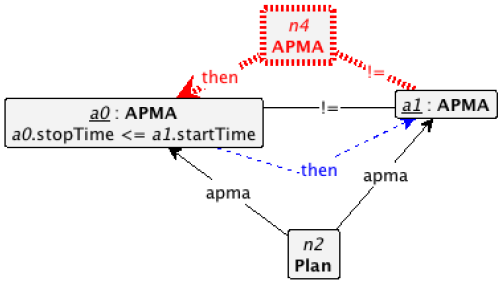
(d) Rule error_duplicate_timepoints



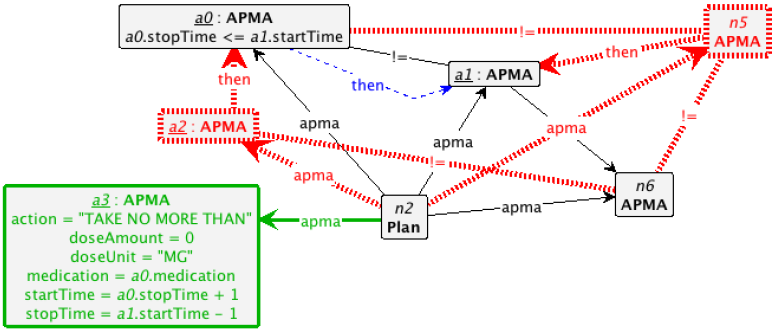
(e) Rule error_frequency_mismatch



(a) Rule plan_completion_1



(b) Rule plan_completion_3



(c) Rule plan_completion_2

Bibliography

[1] To Err Is Human: Building a Safer Health System. Technical report, Institue of Medicine, 2000.

[2] A. Aouat, F. Bendella, and E.A. Deba. Tools of model transformation by graph transformation. In *2012 IEEE 3rd International Conference on Software Engineering and Service Science (ICSESS)*, pages 425–428, June 2012.

[3] Joan S. Ash, Dean F. Sittig, Eric G. Poon, Kenneth Guappone, Emily Campbell, and Richard H. Dykstra. The extent and importance of unintended consequences related to computerized provider order entry. *Journal of the American Medical Informatics Association : JAMIA*, 14(4):415–423, 2007.

[4] Luciano Baresi and Reiko Heckel. Tutorial Introduction to Graph Transformation: A Software Engineering Perspective. In Andrea Corradini, Hartmut Ehrig, Hans-Jrg Kreowski, and Grzegorz Rozenberg, editors, *Graph Transformation*, number 2505 in Lecture Notes in Computer Science, pages 402–429. Springer Berlin Heidelberg, January 2002.

[5] Luciano Baresi and Mauro Pezz. From Graph Transformation to Software Engineering and Back. In Hans-Jrg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *Formal Methods in Software and Systems Modeling*, number 3393 in Lecture Notes in Computer Science, pages 24–37. Springer Berlin Heidelberg, January 2005.

[6] Dorothea Blostein, Hoda Fahmy, and Ann Grbavec. Issues in the practical use of graph rewriting. In Janice Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, number 1073 in Lecture Notes in Computer Science, pages 38–55. Springer Berlin Heidelberg, 1996.

[7] Elizabeth M Borycki. Technology-induced errors: where do they come from and what can we do about them? *Studies In Health Technology And Informatics*, 194:20–26, 2013.

[8] Emily M. Campbell, Dean F. Sittig, Joan S. Ash, Kenneth P. Guappone, and Richard H. Dykstra. Types of unintended consequences related to computerized provider order entry. *Journal of the American Medical Informatics Association : JAMIA*, 13(5):547–556, 2006.

[9] H Ehrig, G Engels, H-J Kreowski, and G Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation: Volume 2: Applications, Languages and Tools*. WORLD SCIENTIFIC, October 1999.

[10] Hartmut Ehrig. *Fundamentals of algebraic graph transformation*. Springer, New York; Berlin, 2006.

[11] Michael A. Fischer, Christine Vogeli, Margaret R. Stedman, Timothy G. Ferris, and Joel S. Weissman. Uptake of electronic prescribing in community-based practices. *Journal of General Internal Medicine*, 23(4):358–363, April 2008.

[12] Carol Friedman and Stephen B. Johnson. Natural language and text processing in biomedicine. In Edward H. Shortliffe MD MACP and James J. Cimino MD FACP, editors, *Biomedical Informatics*, Health Informatics, pages 312–343. Springer New York, 2006.

[13] Amir Hossein Ghamarian, Maarten de Mol, Arend Rensink, Eduardo Zambon, and Maria Zimakova. Modelling and analysis using GROOVE. *International Journal on Software Tools for Technology Transfer*, 14(1):15–40, March 2011.

[14] Stephen Goundrey-Smith. Data support for electronic medicines management. In *Principles of Electronic Prescribing*, Health Informatics, pages 97–115. Springer London, January 2012.

[15] Stephen Goundrey-Smith. Electronic prescribing and future priorities. In *Principles of Electronic Prescribing*, Health Informatics, pages 155–174. Springer London, January 2012.

[16] Stephen Goundrey-Smith. Organization benefits of electronic prescribing. In *Principles of Electronic Prescribing*, Health Informatics, pages 47–67. Springer London, January 2012.

[17] Stephen Goundrey-Smith. Philosophical and social framework of electronic medicines management. In *Principles of Electronic Prescribing*, Health Informatics, pages 1–24. Springer London, January 2012.

[18] Stephen Goundrey-Smith. *Principles of Electronic Prescribing*. Health Informatics. Springer London, January 2012.

[19] Bradi B Granger and Hayden B Bosworth. Medication adherence: emerging use of technology. *Current Opinion in Cardiology July 2011*, 26(4):279–287, 2011.

[20] Yong Y. Han, Joseph A. Carcillo, Shekhar T. Venkataraman, Robert S. B. Clark, R. Scott Watson, Trung C. Nguyen, Hlya Bayir, and Richard A. Orr. Unexpected increased mortality after implementation of a commercially sold computerized physician order entry system. *Pediatrics*, 116(6):1506–1512, December 2005.

[21] C. Henke, M. Tichy, T. Schneider, J. Bocker, and W. Schafer. System Architecture and Risk Management for Autonomous Railway Convoys. In *2008 2nd Annual IEEE Systems Conference*, pages 1–8, April 2008.

[22] Jan Horsky, Gilad J. Kuperman, and Vimla L. Patel. Comprehensive analysis of a medication dosing error related to CPOE. *Journal of the American Medical Informatics Association : JAMIA*, 12(4):377–382, 2005.

[23] Jan Horsky, Shobha Phansalkar, Amrita Desai, Douglas Bell, and Blackford Middleton. Design of decision support interventions for medication prescribing. *International Journal of Medical Informatics*, 82(6):492–503, June 2013.

[24] Rainu Kaushal, Lisa M. Kern, Yolanda Barrn, Jill Quaresimo, and Erika L. Abramson. Electronic prescribing improves medication safety in community-based office practices. *Journal of General Internal Medicine*, 25(6):530–536, June 2010.

[25] Manuel Koch, Luigi V. Mancini, and Francesco Parisi-Presicce. A Graph-based Formalism for RBAC. *ACM Trans. Inf. Syst. Secur.*, 5(3):332–365, August 2002.

[26] Koppel R, Metlay JP, Cohen A, and et al. Role of computerized physician order entry systems in facilitating medication errors. *JAMA*, 293(10):1197–1203, March 2005.

[27] John W McGillicuddy, Mathew J Gregoski, Anna K Weiland, Rebecca A Rock, Brenda M Brunner-Jackson, Sachin K Patel, Beje S Thomas, David J Taber, Kenneth D Chavin, Prabhakar K Baliga, and Frank A Treiber. Mobile Health Medication Adherence and Blood Pressure Control in Renal Transplant Recipients: A Proof-of-Concept Randomized Controlled Trial. *JMIR Research Protocols*, 2(2):e32, August 2013.

[28] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-specific Languages. *ACM Comput. Surv.*, 37(4):316–344, December 2005.

[29] Jose Joaquin Mira, Isabel Navarro, Federico Botella, Fernando Borras, Roberto Nuno-Solinis, Domingo Orozco, Fuencisla Iglesias-Alonso, Pastora Perez-Perez, Susana Lorenzo, and Nuria Toro. A Spanish Pillbox App for Elderly Patients Taking Multiple Medications: Randomized Controlled Trial. *Journal of Medical Internet Research*, 16(4), April 2014.

[30] Olufunmilola K. Odukoya and Michelle A. Chui. e-prescribing: characterisation of patient safety hazards in community pharmacies using a sociotechnical systems approach. *BMJ Quality & Safety*, 22(10):816–825, October 2013.

[31] World Health Organization. Adherence For Long-term Therapies - Evidence For Action. Technical report, 2003.

[32] World Health Organization. Increasing complexity of medical technology and consequences for training and outcome of care: background paper 4, august 2010. 2010. Background paper to "Medical devices: managing the mismatch: an outcome of the Priority Medical Devices project".

[33] Matvey B. Palchuk, Elizabeth A. Fang, Janet M. Cygielnik, Matthew Labreche, Maria Shubina, Harley Z. Ramelson, Claus Hamann, Carol Broverman, Jonathan S. Einbinder, and Alexander Turchin. An unintended consequence of electronic prescriptions: prevalence and impact of internal discrepancies. *Journal of the American Medical Informatics Association*, 17(4):472–476, July 2010.

[34] Terence Parr. *The Definitive ANTLR 4 Reference*. The Pragmatic Bookshelf, 2 edition, September 2014.

[35] Terence Parr and Kathleen Fisher. LL(*): The Foundation of the ANTLR Parser Generator. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 425–436, New York, NY, USA, 2011. ACM.

[36] John L. Pfaltz and Azriel Rosenfeld. Web Grammars. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, IJCAI'69, pages 609–619, San Francisco, CA, USA, 1969. Morgan Kaufmann Publishers Inc.

[37] Gurprit Kaur Randhawa. *Evaluating the adoption of electronic prescribing in primary care*. Thesis, 2013.

[38] Ulrike Ranger and Erhard Weinell. The Graph Rewriting Language and Environment PROGRES. In Andy Schrr, Manfred Nagl, and Albert Zndorf, editors, *Applications of Graph Transformations with Industrial Relevance*, number 5088 in Lecture Notes in Computer Science, pages 575–576. Springer Berlin Heidelberg, 2008.

[39] Michael A. Rapoff. Definitions of adherence, types of adherence problems, and adherence rates. In *Adherence to Pediatric Medical Regimens*, Issues in Clinical Child Psychology, pages 1–31. Springer US, January 2010.

[40] Arend Rensink. The GROOVE Simulator: A Tool for State Space Generation. In John L. Pfaltz, Manfred Nagl, and Boris Bhlen, editors, *Applications of Graph Transformations with Industrial Relevance*, number 3062 in Lecture Notes in Computer Science, pages 479–485. Springer Berlin Heidelberg, 2004.

[41] Grzegorz Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation*. World Scientific, Singapore ; New Jersey, 1997.

[42] Andy Schrr and Felix Klar. 15 Years of Triple Graph Grammars. In Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *Graph Transformations*, number 5214 in Lecture Notes in Computer Science, pages 411–425. Springer Berlin Heidelberg, 2008.

[43] Simon Diemert, Kirk Richardson, Paul Hunter, Jens H Weber, and Morgan Price. SmartMed: A Medication Management System to Improve Adherence. In *Proc. of Information Technology and Communications in Health (ITCH)*, Victoria, BC, Canada, March 2015. IOS Press.

[44] Sarah P Slight, Rachel Howard, Maisoon Ghaleb, Nick Barber, Bryony Dean Franklin, and Anthony J Avery. The causes of prescribing errors in english general practices: a qualitative study. *The British Journal of General Practice*, 63(615):e713–e720, October 2013.

[45] Giovanni Toffetti and Mauro Pezz. Graph transformations and software engineering: Success stories and lost chances. *Journal of Visual Languages & Computing*, 24(3):207–217, June 2013.

[46] Pei-Hsuan Tsai, Tsung-Yen Chen, Chi-Ren Yu, Chi-Sheng Shih, and J.W.S. Liu. Smart medication dispenser: Design, architecture and implementation. *IEEE Systems Journal*, 5(1):99–110, March 2011.

[47] A. Turchin, A. Sawarkar, Y. A. Dementieva, E. Breydo, and H. Ramelson. Effect of EHR user interface changes on internal prescription discrepancies. *Applied Clinical Informatics*, 5(3):708–720, 2014.

[48] Upkar Varshney. Smart medication management system and multiple interventions for medication adherence. *Decision Support Systems*, 55(2):538–551, May 2013.

[49] Jon J. Vlasnik, Sherry L. Aliotta, and Bonnie DeLor. Medication adherence: Factors influencing compliance with prescribed medication plans. *The Case Manager*, 16(2):47–51, 2005.

[50] Philip S. Wang, Joshua S. Benner, Robert J. Glynn, Wolfgang C. Winkelmayer, Helen Mogun, and Jerry Avorn. How well do patients report noncompliance with antihypertensive medications?: a comparison of self-report versus filled prescriptions. *Pharmacoepidemiology and Drug Safety*, 13(1):11–19, January 2004.

[51] Robert L. Wears and Nancy G. Leveson. safeware: Safety-critical computing and health care information technology. In Kerm Henriksen, James B. Battles, Margaret A. Keyes, and Mary L. Grady, editors, *Advances in Patient Safety: New Directions and Alternative Approaches (Vol. 4: Technology and Medication Safety)*, Advances in Patient Safety. Agency for Healthcare Research and Quality (US), Rockville (MD), 2008.

[52] Han-Chun Yeh, Pi-Cheng Hsiu, Chi-Sheng Shih, Pei-Hsuan Tsai, and J.W.-S. Liu. APAMAT: A prescription algebra for medication authoring tool. In *IEEE*

International Conference on Systems, Man and Cybernetics, 2006. SMC '06, volume 5, pages 4284–4291, October 2006.