Using Graph Transformations for Formalizing Prescriptions and Monitoring Adherence

Jens H. Weber [1,2(✉)], Simon Diemert [1], and Morgan Price [1,2]

[1] Department of Computer Science, University of Victoria, Victoria, Canada
jens@uvic.ca
[2] Department of Family Practice, University of British Columbia, Vancouver, BC, Canada

Abstract. Medication prescriptions are an important class of medical intervention orders. Their complexity ranges widely, depending on the nature of the patient's condition and the prescribed substance(s). In today's IT supported clinical environments, prescriptions are often authored electronically. Patient adherence to the prescribed medication regimen is a key determinant for the outcome of the intervention. Recently, an increasing number of information technologies are entering the consumer market with a goal to assist patients with adhering to their prescriptions. The effectiveness (and safety) of these technologies is limited to simplistic cases, however, because of the lack of a precise semantics for more complex prescription orders. To close this gap, we present an approach to formalize the meaning of medication prescriptions based on a graph-transformation system. This allows for more complex and variable prescriptions to be semantically coded and their adherence to be automatically monitored. Our work has been implemented within a prototypical prescribing tool and validated with domain experts.

1 Introduction

Medications are an important form of medical interventions. In modern health care systems, medications are often and increasingly prescribed using software-supported clinical information systems, commonly referred to as computerized provider order entry (CPOE) systems. The user interfaces of modern CPOE systems are typically partially structured (form-based), but also allow for unstructured information entry (free text) in order to provide flexibility for complex prescription orders and patient-specific constraints. The health outcome and safety of prescriptions ordered in primary care relies to a large degree on clarity of the instructions and the patient's ability (and willingness) to adhere to the prescribed medication regimen. The World Health Organization (WHO) has identified poor adherence to medication regimens as a world-wide problem "of striking magnitude" [3].

Recent developments in the consumer health market have created a rapidly expanding array of technologies with the goal to help patients with adhering to their medication regimens [6,15]. The effectiveness (and safety) of these technologies is limited by their ability to correctly capture and interpret prescription

F. Parisi-Presicce and B. Westfechtel (Eds.): ICGT 2015, LNCS 9151, pp. 205–220, 2015.
DOI: 10.1007/978-3-319-21145-9_13

orders. Unfortunately, current e-prescribing systems lack a precise, formalized semantics for prescription orders, which may lead to ambiguities and misunderstandings on how to interpret complex medication plans. This paper presents an approach to address this limitation. We define a domain specific language for writing electronic prescriptions and a graph transformation system to precisely specify prescription semantics. The approach has been implemented in a prototypical tool that connects a physician's e-prescribing system with adherence monitoring devices deployed in the patient's personal environment.

The rest of this paper is structured as follows. The following section provides the reader with a more detailed description of the health care process targeted in this paper, i.e., the medication management process and specifically the role of prescription in that process. We discuss related work in Sect. 3. Section 4 lays out the proposed graph-transformation-based method to formalize and interpret electronic prescriptions. We evaluate our approach in Sect. 5 and offer concluding remarks in Sect. 6.

2 Medication Management Process

Medication management (MM) is a complex and multi-faceted concern with many variations depending on the particular health context of a patient's condition and the organization of the health care system that supports their treatment. Given this complexity, the MM process model described here has limited applicability and is not meant to be comprehensive. It specifically applies to medications managed in primary and ambulatory care (outpatient) scenarios and models the four main steps involved in the patient's treatment cycle, namely (1) *testing* the patient's condition, (2) *prescribing* a medication intervention (after reviewing the patient's chart, e.g., for allergies, interactions with other existing medications, etc.), (3) *dispensing* the medication (at a pharmacy), and (4) *administering* the medication (usually at home). These steps are usually iterated multiple times in order to iteratively control the patient's health condition, particularly in the context of chronic disease management.

Figure 1 depicts an overview of this process with solid arrows representing the typical information flows and dashed arrows representing optional information flows, i.e., information flows that benefit medication management but may not always be present. For example, the communication of a medication dispensation event from the pharmacy back to the clinician can improve patient safety (e.g., patients may have forgotten, lost or may want to avoid the cost of filling prescriptions), *if* the system is set up to support this flow. Similarly, the communication of information about the patient's adherence to a medication regimen (called *adherence trace* in Fig. 1) helps the clinician understand to what degree the planned intervention was actually performed. Such an adherence trace can be generated based on automated IT devices embedded in the patient's home (e.g., smart pill bottles [6]) or it can be based on manual methods, e.g., patient's recollection or "pill counts".

A prerequisite for automated adherence tracing (and indeed also for accurate manual adherence and tracing) is a precise, unambiguous understanding of the

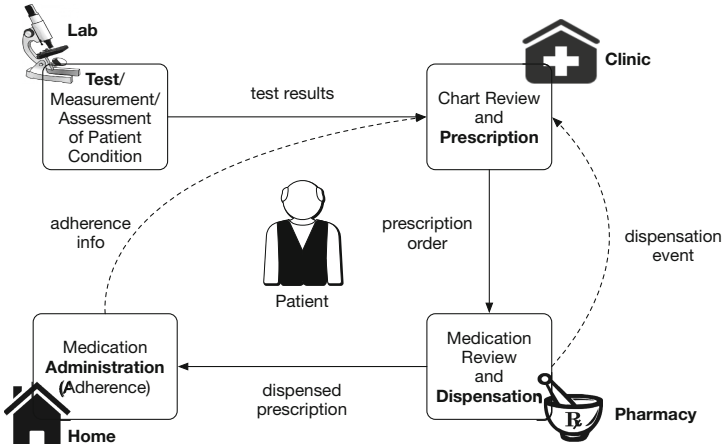

Fig. 1. Medication management process

meaning of prescriptions. Unfortunately, current e-prescribing (CPOE) systems do not commonly utilize prescribing languages with formally defined semantics. While some primitives of a prescription order may be structured and can be considered unambiguously encoded (e.g., the prescribed substance is usually encoded with a controlled vocabulary), other prescription elements are formulated in representations that lack formal semantics. It is this problem of how to define a prescription language with machine-interpretable semantics (allowing for automated adherence tracking) that we address in this paper.

To provide the reader with an appreciation of a typical e-prescribing interface, Fig. 2 depicts the medication order screen of the OSCAR Electronic Medical Record (EMR) software (version 12.1), a software product in use by over two thousand primary care physicians in Canada [8]. The user interface provides for semi-structured order entry. Medication substances are looked up from a database of known drugs (top field in Fig. 2). The next field (entitled "*Instructions*") provides the clinician with a way to textually input medication instructions (such as quantity, dose, strength, timing etc.). If the clinician uses certain conventions or keywords when entering these instructions (cf. pop-up help displayed on the right side of CPOE screen), the EMR software is able to extract certain pieces of information and automatically populate some or all of the entry fields below (e.g., quantity, repeats, route).

From a practical perspective, it is important to note that oftentimes prescription orders are (purposefully) underspecified when entered and submitted by the clinician. For example, physicians often prescribe generic substances (rather than drug brand names) and doses (rather than pill sizes and quantities). This leaves flexibility to the pharmacist to select suitable brands, pill sizes etc. based on the pharmacy's inventory and other considerations, such as patient preferences, insurance coverage and medication cost. This means that the medication process to be adhered to from a patient's point of view is often "refined" at the pharmacy into an actionable task plan, e.g., "*take two pills a day ...*" rather than

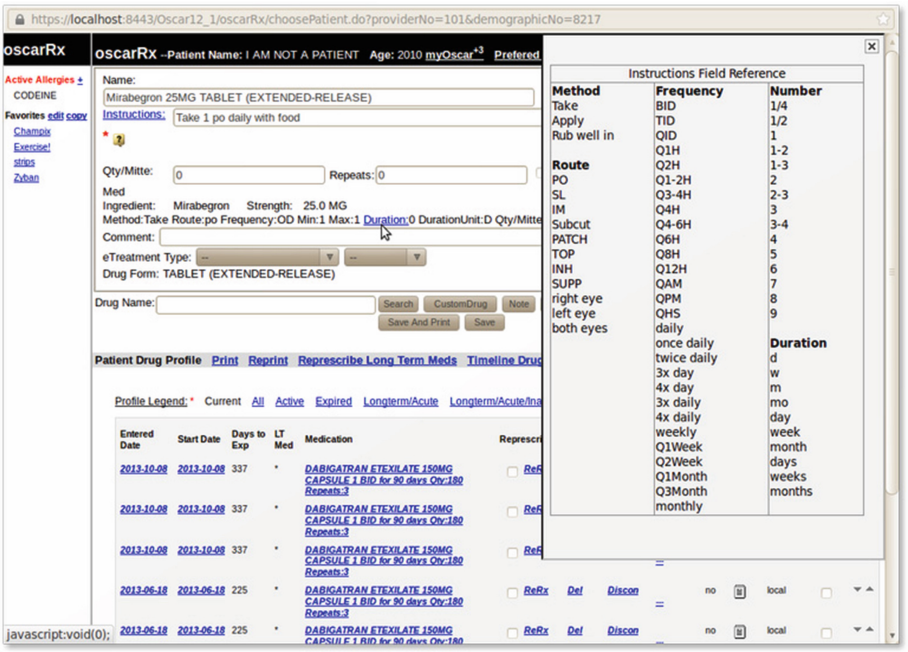

Fig. 2. Example CPOE prescription order screen from OSCAR EMR [8]

"*take 200 mg a day ...*". For simplicity, our current method and tool implementation does not explicitly consider this two-step "refinement" process between clinic and pharmacy. In other words, we treat authoring the prescription as a single abstract process step. We note, however, that our method can naturally be extended to account for this two-step prescription authoring process.

3 Related Work

Yeh et al. developed a machine-readable medication schedule specification (MSS) based on a prescription algebra called APAMAT (A Prescription Algebra for Medication Authoring Tool) [17]. A main objective of their tool is to validate multiple prescriptions for potentially dangerous interactions (drug-drug interactions or drug-allergy interactions). If no interactions are found, APAMAT creates a schedule that can be used for adherence monitoring. Yeh et al. define the structure of their algebra (and the grammar of their corresponding domain specific input language) formally, but their presentation lacks a formal definition of the semantic concepts.

Varshney presents the requirements and the conceptual design of a smart medication management system (SMMS) for improving adherence [15]. He discusses a theoretical framework for adherence to medication regimes and a framework for evaluating the effectiveness of any SMMS. Diemert et al. present SmartMed, a prototype medication adherence system based on a smart, mobile medication

container ("pill bottle") capable of communicating to cloud-based information system [6]. SmartMed has been developed independently but implements several of the design features proposed for SMMS by Varshney. The prescription language and graph transformation-based adherence monitoring approach presented in this paper has been developed for the SmartMed system.

Beyond the context of medication prescriptions, Yan et al. have conducted research on formalizing a notion of adherence to general clinical work flows [16]. They use the Business Process Modeling Notation (BPMN) to specify desired work flows and evaluate clinical adherence to these models based on captured activity traces. While this work is related to ours, the kinds of phenomena modelled in Yan et al.'s work do not align well with the problem of medication adherence, as framed in this paper. For example, there is no consideration for time, medication substance and strength in Yan et al.'s approach.

The research presented in this paper is related to the general research area on domain specific languages (DSL) [10,17]. Model-based approaches are popular among the various approaches proposed for developing DSL-based systems [2]. An important aspect in the development of DSL-based systems is the formal definition of the language semantics. Popular approaches use mappings of the DSL into precisely defined mathematical formalisms or utilize rewrite rule systems [4]. Typed graphs and graph transformations have been used extensively for defining semantic models for DSLs [2,7]. Graph transformations are defined as rules where the left-hand side describes the structure of a subgraph to be matched in a given instance graph and a right-hand side which replaces the matched subgraph upon application of the rule. Different notations and tools have been developed to support the specification and execution of graph transformations, e.g., [1,5,13,14]. The specific notation and tool used in our application is GROOVE [13].

4 A GT-Based Method to Formalize Prescriptions

Formalizing a language for medication prescriptions requires two main tasks, namely (1) the definition of an *interface language* to be used by clinicians to author prescriptions, and (2) the transformation of prescriptions authored in this interface language to a *formal model* that defines the semantics of what has been prescribed. The interface language can be textual, visual (form-based), or hybrid, as in the case of OSCAR's CPOE module (Fig. 2). Without loss of generality, we elected to develop a textual interface language for our prototype. If visual or hybrid interface languages are preferred they can be translated into our textual representation.

We choose a graph transformation system (GTS) as a way to formally model the semantics of prescription orders. Of course, alternative formal methods could have been selected, e.g., Petri nets [11], temporal action logics [9], and any other formalism capable of modeling processes. Indeed we wrote some specifications using Petri nets, and TLA+ process models, and the Z notation [12] before settling on the GTS approach presented here. One reason for selecting the GTS approach was that we could use graph transformations to describe the mapping if

the interface language to the semantic model as well as the interpretation of that semantic model. TLA+, Z and Petri nets are primarily specification formalisms. They do not lend themselves well to defining the mapping between a concrete textual DSL and a formal specification. A second reason for selecting GTS was our need to communicate the specified semantics with domain experts that were not trained in formal methods. Graphs and graph transformation rules turned out to be much more accessible for this purpose. A third reason for our choice was the available tool support, which in the case of GTS tools (and Petri net tools) encompassed specification as well as model based execution, while the tooling for other formalisms (such as TLA+ and Z) focuses mainly on specification and verification.

Figure 3 provides an overview of our method in form of a FlowChart. Prescription orders are parsed from clinical input using a textual interface language (DSL). The DSL parsing process populates a graph model, referred to as the "Rx Graph Model" in Fig. 3. Since the interface language for prescription orders may contain complex primitives, our next step compiles the Rx Graph to a simpler representation of the prescribed medication actions, referred to as the APMA (Atomic Prescribed Medication Action) Graph (cf. "Rx Compiler" process). This compilation step is performed by a graph transformation system (GTS). The compiler also validates static semantic constraints of our prescription language.

The semantics of the APMA Graph is defined by a GTS that relates the APMA Graph model to another graph model that captures actual medication administration events (the "Administration Graph Model") in Fig. 3.

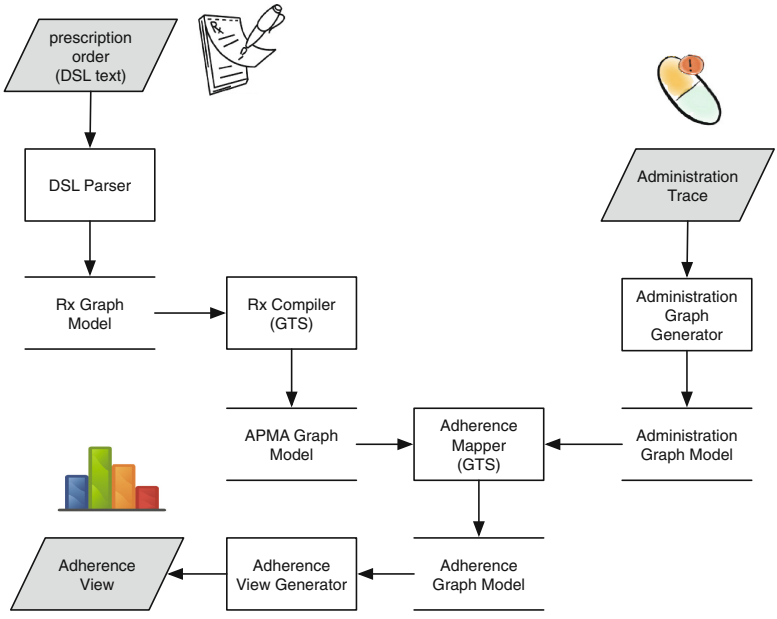

Fig. 3. Method for formalizing prescriptions and monitoring adherence

The Administration Graph is populated by sourcing medication administration events from smart devices and sensors embedded within the patient's environment, e.g., a smart pill bottle [6]. Finally, our system design includes an *Adherence View Generator* that provides medication adherence information back to the clinician and/or the patient based on the data in the Adherence Graph. While the Adherence View Generator is not a core topic of this paper (it is not required for defining the semantics of prescription orders), it is an important component of the overall application, as it produces the adherence information in the medication management process (cf. Fig. 1) and should thus be mentioned here for completeness.

Now that we have provided an overview of our approach, we will describe each component in more detail in the next subsections.

4.1 Interface Language

We developed the grammar of a textual interface language for creating medication prescriptions. The language is based on a literature survey as well as on input from a domain expert. (One of the co-authors is a primary care physician.) As previously mentioned, prescriptions can be complex; to simplify our task, the current version of the language focuses on those aspects of a medication prescription than can be tracked with typical administration monitoring devices available to patients in the community. These aspects are related to the medication substance, the timing and the dosing of the medication. Other aspects that are harder to monitor have been left for future extension, e.g., the administration *route* (e.g., oral, topical, rectal, etc.).

Below are five example prescription orders of different complexity written in the developed interface language. Order 1 and 2 are simple and do not require further explanation. Order 3 uses more complex timing (weekdays as well as time of the day) and also varies the medication dose based on the time of the day. Order 4 illustrates a prescription that uses two medications in strict sequence. Order 5 varies the medication dose by applying a titrating process. Moreover, it requires the patient to repeat the titrating process once (for a total of 20 days).

1. `Take chloronapam 80 mg once daily for 60 days`
2. `Take adhdhesin 150 mg twice daily (8, 20) for 10 days` (specific times of day: 8 AM and 8 PM)
3. `Take stalacillin (10 mg, 20 mg) three times weekly (1, 3, 5) at (8, 20) for 10 weeks` (specific days and times, varying doses)
4. `Take chordazine 75 mg daily for 7 days then take chordazine 150 mg for 28 days` (sequential medication)
5. `Take planazipine titrate down from 50 mg to 0 mg by 10 mg per two days once daily for 10 days` (titrated medication increase or decrease dose at each interval)

We note that the interface language below has been designed primarily for expressiveness and as a vehicle to feed our proof-of-concepts prototype. We have

not studied it from a usability/user experience perspective. Indeed, textual interface languages used in current CPOE systems often try to minimize verbosity and make use of abbreviations and shorthand codes (cf. right-hand side of Fig. 2 as an example). Usability research on prescription interface languages is subject of ongoing and future work in our lab, but not the focus of this paper.

4.2 Rx Graph Model

Prescriptions written in the interface language are parsed to populate a typed graph model, referred to as the "Rx Graph" in Fig. 3. Figure 4 shows the Rx Graph representations for the sample prescriptions 3 and 5 above. The corresponding graph model (type graph) is given in Fig. 5, using GROOVE notation [13].

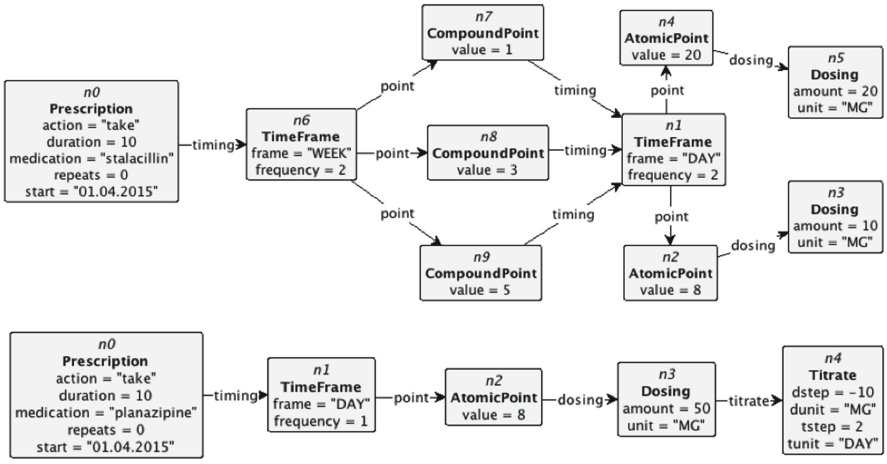

Fig. 4. Rx Graph representation for sample prescription orders 3 and 5

Prescriptions are represented as attributed graph nodes specifying the type of the action, the medication substance, a start time, a duration and a number of "repeats". The start time is automatically initialized to the time of prescribing (dispensing) the medication, unless otherwise explicitly specified in the textual interface language. The timing of a prescription is specified in a recursive graph structure defined by a *time frame*, which contains one or many *time points*. Consider our prescription order 3 from above as an example. The top part of Fig. 4 shows that the timing of that order consists of a weekly time frame with two time points (with values 1 and 4, representing Monday and Thursday, respectively). Each of these time points is referred to as *compound* as it in turn represents a time frame (day) with two time points (8am and 8pm). The latter time points are not further refined by time frames, i.e., they are referred to as *atomic* rather than compound. Atomic time points are related to actual dosing actions, represented by instances of "Dosing" graph notes.

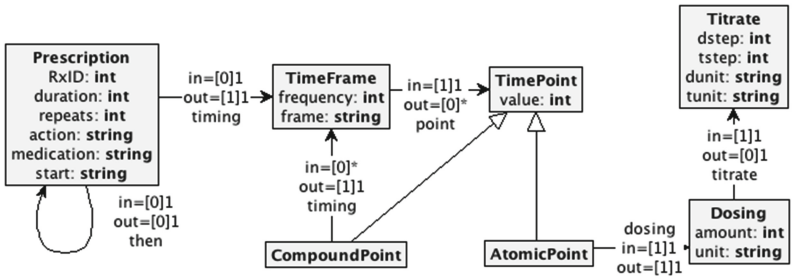

Fig. 5. Rx Graph Model (type graph)

4.3 Rx Graph Compilation

The compilation of the Rx Graph into the APMA Graph is implemented as a graph transformation system in GROOVE [13]. GROOVE was selected out of a set of five GTS tools listed on Wikipedia's page on graph rewriting page as "*domain neutral*"[1]. We were particularly interested in a tool that provided support for formal verifications of the GTS system (e.g., confluence) as well as code generation for Java. GROOVE as well as AGG [14] met these requirements and were considered for closer evaluation in our project. We eventually decided to select GROOVE since the tool provides for a more compact representation of transformation rules, i.e., a rule's left-hand and right-hand sides are folded into a single graph representation.

The target model for the compilation (APMA Graph) is simple. It merely consists of a set of atomic medication actions (respectively *in*actions) that are planned for absolute time intervals. Its type model is shown on the left hand side of Fig. 6 (node types *Prescription* and *APMA*). The compilation process consists of five main phases:

1. **Static Semantic Validation.** A set of graph rules are applied prior to further processing to validate static semantic properties of prescription orders, e.g., to ensure that the specified frequency aligns with the specified medication time points. Figure 7 shows a corresponding graph test in GROOVE notation. The check counts the number of TimePoints connected to a TimeFrame and compares it to the frequency attribute specified for the TimeFrame. Under this graph rule, the following prescription order would be found invalid for example: "`take aspirin 81 mg once daily (8, 20) for 10 days`".
2. **Time Unrolling.** In this phase, the compiler computes absolute time points for planned medication actions based on the timing description of the prescription order. Iterations and repetitions are "unrolled" and abstract references to time frames (e.g., "daily", "Monday", etc.) are replaced with absolute times (using *Unix time* (en.wikipedia.org/wiki/Unix_time) for simplicity). Figure 8 presents a graph transformation rule that unrolls "day" time frames. Analogous transformation rules exists for months and weeks. Obviously, this compilation step

[1] http://en.wikipedia.org/wiki/Graph_rewriting.

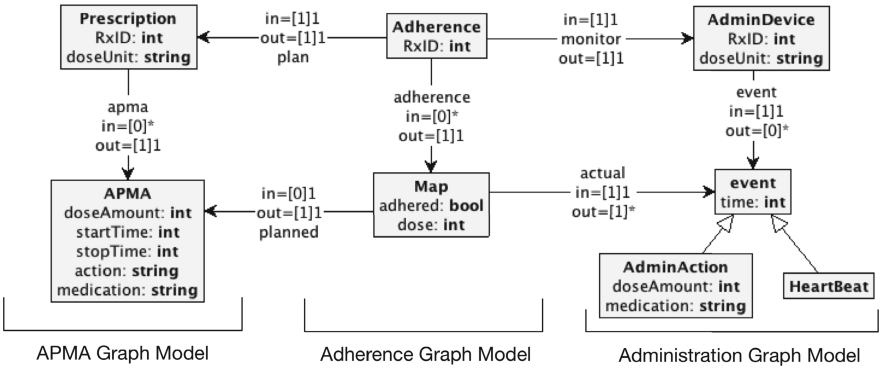

Fig. 6. Monitoring graph triplet (type graph)

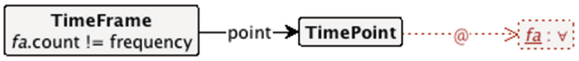

Fig. 7. Example graph tests to validate static semantic constraints

may create unreasonable precision. For example, a prescription to take a certain medication "next Monday" will be compiled to a concrete planned medication time point at the resolution of seconds (Unix time) at noon of the following Monday. This issue is addressed in the following compilation step.

3. **Temporal Unsharpening.** In this phase, absolute time points generated previously are replaced by intervals. This is necessary because of the above mentioned issue of unreasonable precision. The width of the generated intervals created depend on the level of precision in the original prescription order. If for example, a medication action is prescribed at the level of a "day", the generated interval extends 43200 seconds to both sides of the previously generated time point.
4. **Plan Completion.** The medication plan generated so far is *partial* in the sense that it defines required actions to happen at specific times (intervals), but it does not specify whether medication administration actions are permitted outside these intervals. (This may sometimes be the case, for example in pain medication prescriptions that specify certain minimum doses but allow patients to add doses "as needed".) Our current interface language does not yet allow clinicians to specify "as needed" options. However, our graph models have been designed to incorporate this aspect at a later time. The objective of the last compilation phase is to create a *total* medication plan from the *partial* plan generated thus far, by filling in planned intervals of prohibited medication actions between planned intervals of planned medication actions. In other words, we assume that (unless otherwise specified) clinicians do not intend patients to take their prescribed medications outside the prescribed times. (Of course, this is a simplifying assumption. We will discuss this limitation in the last section of this paper.)

5. **Dose Unit Harmonization.** The final step in the compilation harmonizes the dose unit information. While the interface language (and the Rx Graph model allows different dose units to be used in authoring a prescription (e.g., milligrams, grams), the target APMA model uses a single dose unit per prescription (cf. "doseunit" attribute of node type "Prescription" in Fig. 6).

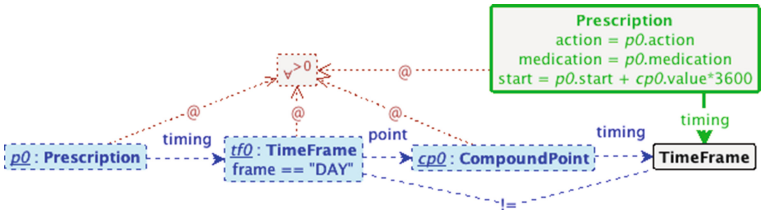

Fig. 8. Example compilation rule for prescription "time unrolling"

Figure 9 shows an excerpt of an APMA graph generated for prescription 3 in our list of examples above.

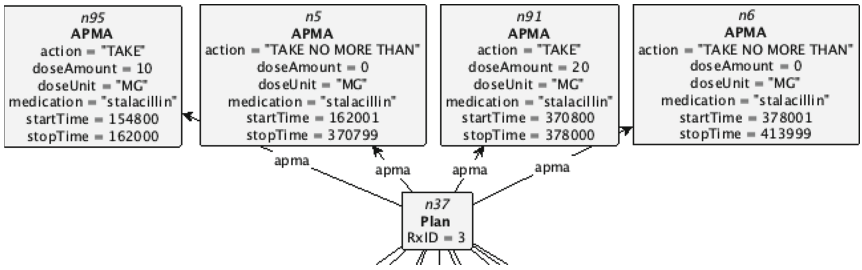

Fig. 9. Compilation result (APMA Graph) for our prescription example 3

4.4 Adherence Tracking

We use graph transformations for specifying the dynamic semantics of the APMA Graph model. The right hand side of Fig. 6 shows the Administration Graph model, which is used to capture actual medication administration events, as emitted by a persons, a smart medication administration tool or similar monitoring device embedded with the patient, e.g., a smart pill bottle [6]. The monitoring device is capable of emitting two types of events: (1) a medication administration event (which is accompanied by information about the administered dose) and (2) a heartbeat. The heartbeat is emitted at a (customizable) regular interval (e.g., once a day) to ensure that the monitoring device is still functioning. It is also used in the medication adherence tracking process. We assume that monitoring devices are uniquely associated with prescriptions at the time of dispensation.

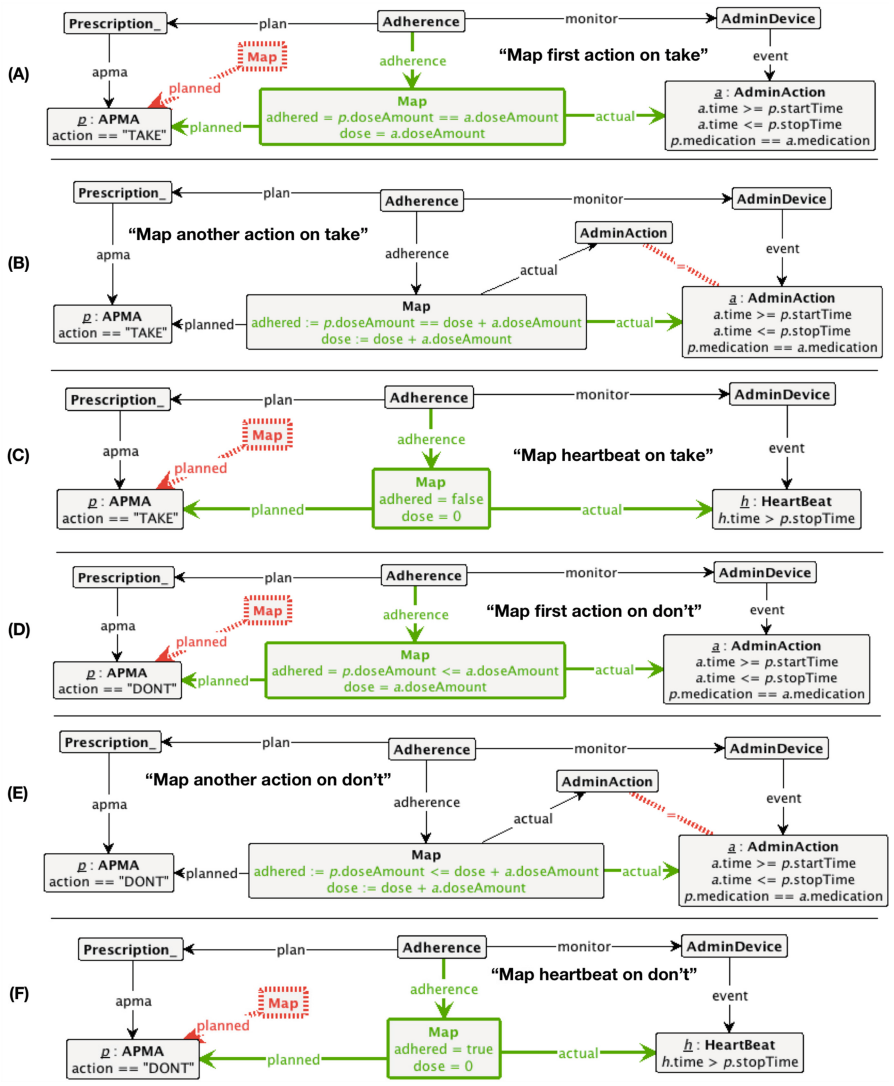

Fig. 10. Transformation rules for creating the Adherence Graph

Every time a monitoring device emits an event, that event (administration or heartbeat) is recorded in the Administration Graph. Figure 10 shows six graph transformations that generate the Adherence Graph, based on events recorded from the monitoring device and compiled prescription plan in the APMA Graph. The Adherence Graph consists of instances of mapping nodes (*"Map"*) that associate planned (*in*)actions with actual events. Positive adherence is marked by a Boolean attribute *"adhered"* that is recorded as `true`.

The first transformation rule in Fig. 10 covers the case when a medication administration event is recorded during a time when the patient was asked to

take her medication. In this case, correct adherence to the prescription plan depends on whether the correct dose was administered. Now the patient may actually administer multiple doses during a planned prescription time frame (e.g., she may open the smart pill bottle twice to take one pill each time). This action will lead to two medication administration events being recorded. The overall dose administered during the planned time interval should be computed as the total of all administered doses. This function is performed by the second rule in Fig. 10. (Note that the overall dose is kept in the Adherence Graph.) The third rule (Rule C) records a non-adherence case when a heartbeat is received after expiration of a time interval where an administration action was planned (and none was recorded). Note that the absence of a recorded administration event is guaranteed by the rule's negative application condition (NAC).

Rules D-F work analogously to Rules A-C but consider planned periods of medication inaction (as indicated by the "DON'T" value of the APMA's action attribute). Note that the semantics we are defining here are not a simple prohibition of any medication. Rather we define a "DON'T" action to mean "*don't take more than*". This semantics provides more flexibility and expressiveness for prescription orders. For example, it may be the case that a physician allows patients to take more than the prescribed medication "*if needed*" - but not more than a certain maximum. Rules D and E define these semantics formally. Finally, Rule F handles the situation where a heartbeat is mapped to a planned period of inaction, resulting in positive adherence for that period.

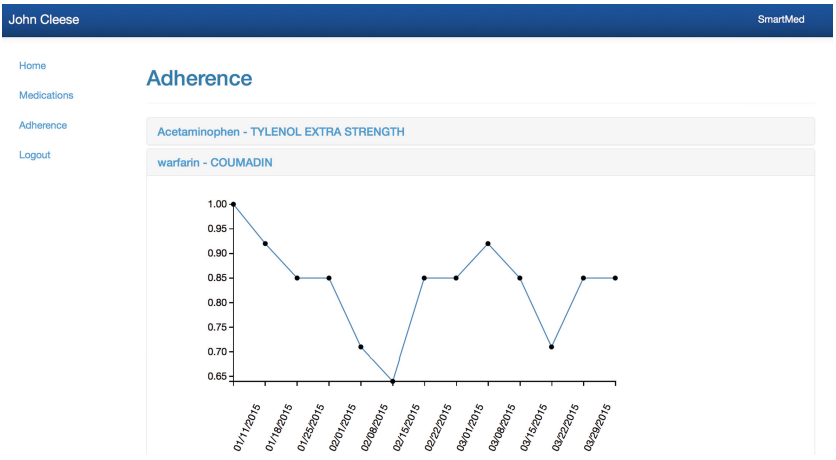

Fig. 11. Screenshot of prototype medication adherence view.

The information recorded in the Adherence Graph can be aggregated to provide end-user specific view points for reflecting on medication adherence. As presented in our overall MM process model (Fig. 1), such viewpoints may be provided for physicians, but they may also be of interest for patients or their

informal circle of care (family members). Figure 11 shows a screen shot of a prototype adherence view we developed for inclusion in a physician's clinical information system (EMR). The view was created based on a simple scoring system, computing the ratio of "*Map*" nodes in the Adherence graph that indicate positive adherence divided by the total count of these nodes over a selected time resolution. More sophisticated and differentiated adherence view metrics could be constructed taking in consideration the difference between planned and actual doses and/or the difference between planned and actual timings. This is subject to future work and not the focus of this paper.

5 Evaluation

The development of a comprehensive, formalized medication prescription and automated adherence monitoring system is a complex challenge. Our system has several known limitations and requires further extension and validation before we can be confident that it is fit for use in practice. As we pointed out earlier, we did not engineer our current interface language (DSL) with an eye to usability and efficiency. Any actual implementation of our system in practice will have to connect the Rx Graph model with an interface language that has been optimized (and validated) for usability.

More important for the topic of this paper is the expressiveness of the language and the associated graph models. The expressiveness of the current language/models have been validated by one domain expert (physician and co-author). The choice of a GTS formalism modelling language semantics has proven instrumental in making the semantic formalization accessible to collaborators that have not been trained in formal methods. Our DSL language and graph model is considered sufficient to capture a large portion of typical prescriptions written for primary care medications. Still, we have so far ignored the aspect of drug interactions and directions on how to recover from non-adherence. For example, patients who are on multiple prescriptions may be asked to never take two (or more) of their drugs at the same time. Moreover, patients who have failed to adhere to a planned medication dose may be asked to perform different actions for recovery, depending on their condition and the nature of the medication, e.g., they may be asked to "skip" the dose, to "double up", or to "take ASAP - but delay the next one". A more expressive prescription language would allow providers to specify these additional constraints.

Another limitation of our current system is that it does not distinguish between the physician's act of writing the initial (loosely constrained) prescription and the pharmacist's subsequent act of "refining" the prescription (constraining it further). Still, making a distinction between the act of prescribing and the act of dispensing does not require an extension to the theoretical framework of our approach. It merely requires the development of another set of graph transformation rules to be used for specifying the permissible refinement actions that can be performed by pharmacists.

From a theoretical, language-engineering point of view our graph transformation-based approach provides a *partial* formal semantic definition of

our interface language (DSL). However, it does not currently guarantee that all well-formed sentences in our interface language have a unique, valid interpretation in terms of an interpretable APMA Graph. The desired property of a *total* formal semantic definition of our interface language for prescriptions requires a proof that the graph transformation system is confluent and terminating (convergent) for all valid inputs. We utilized GROOVE's state space exploration tool to check for convergence of our rule system for all valid inputs. Our future work will be on constructing convergence proofs for all possible sentences of our language.

Considering the adherence tracking rules, we have made several simplifying assumptions in our current system. First of all, we require actual measured dosage to be *exactly* equal to the planned dosage to be accepted for positive adherence. This assumption may be fine for coarse granular units such as "pills" but will be unrealistic for other unit measures, e.g., milligrams. Some means of "unsharpening" should be created to provide a more realistic mapping. Secondly, we currently compile the prescription into a fixed medication plan that does not allow readjustments, e.g., in order to react to slippage. It would be more realistic to be able to dynamically shift the plan in case there is a delay. For example, if the patient filled a prescription and then went on a business trip, forgetting the drugs at home. In this case, they would likely start taking the medication after their return. In some cases, such a delay may be permissible. Our current system does not implement such a function, but it is possible to "shift" the timing in the medication plan (APMA) graph accordingly and rerun the adherence mapping rules to calculate a new adherence graph in such cases.

6 Conclusions and Future Work

Medication non-adherence is a significant health problem world-wide [3]. Health information technologies, consumer health apps and the emerging health Internet of Things (IoT) provide opportunities to pro-actively monitor (and improve) medication adherence [15]. Adherence is a complex process that can be better understood now that we have these new ways of feasibly measuring adherence. Automated medication adherence monitoring requires a formalized, machine-interpretable language for writing and representing prescription orders. Graph transformation systems are a suitable formalism for developing such a language. The graph transformation-based medication management (MM) system discussed in this paper is part of a larger project initiative that has also developed a prototype "smart pill bottle", which is capable of wirelessly emitting medication administration events to cloud-based systems [6]. We are currently planning a small-scale pilot deployment of the MM system to gain feedback on the current design prior implementing more advanced features, such as the extensions mentioned in the previous section.

References

1. Amelunxen, C., Königs, A., Rötschke, T., Schürr, A.: MOFLON: a standard-compliant metamodeling framework with graph transformations. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 361–375. Springer, Heidelberg (2006)
2. Andrés, F.P., de Lara, J., Guerra, E.: Domain specific languages with graphical and textual views. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) AGTIVE 2007. LNCS, vol. 5088, pp. 82–97. Springer, Heidelberg (2008)
3. Brown, M.T., Bussell, J.K.: Medication adherence: who cares? In: Mayo Clinic Proceedings, vol. 86, pp. 304–314. Elsevier (2011)
4. Bryant, B.R., Gray, J., et al.: Challenges and directions in formalizing the semantics of modeling languages. Comp. Sci. Inform. Sys. **8**(2), 225–253 (2011)
5. de Lara, J., Vangheluwe, H.: AToM^3: a tool for multi-formalism and meta-modelling. In: Kutsche, R.-D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, pp. 174–188. Springer, Heidelberg (2002)
6. Diemert, S., Richardson, K., et al.: SmartMed: a medication management system to improve adherence. Stud. Health Technol. Inform. **208**, 125–130 (2015)
7. Heckel, R.: Graph transformation in a nutshell. ENTCS **148**(1), 187–198 (2006)
8. Ruttan, J.: OSCAR. In: The Architecture of Open Source Applications. Structure, Scale and a Few More Fearless Hacks, vol. II (2012)
9. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co. Inc., Amsterdam (2002)
10. Mernik, M., Heering, J.J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. (CSUR) **37**(4), 316–344 (2005)
11. Peterson, J.L.: Petri nets. ACM Comput. Surv. (CSUR) **9**(3), 223–252 (1977)
12. Potter, B., Till, D., Sinclair, J.: An introduction to formal specification and Z. Prentice Hall PTR, Upper Saddle River (1996)
13. Rensink, A.: The GROOVE simulator: a tool for state space generation. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 479–485. Springer, Heidelberg (2004)
14. Taentzer, G.: AGG: a graph transformation environment for modeling and validation of software. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 446–453. Springer, Heidelberg (2004)
15. Varshney, U.: Smart medication management system and multiple interventions for medication adherence. Decis. Support Syst. **55**(2), 538–551 (2013)
16. Yan, H., Van Gorp, P., et al.: Analyzing conformance to clinical protocols involving advanced synchronizations. In: IEEE Conference on Bioinformatics and Biomedicine (2013)
17. Yeh, H.-C., Hsiu, P.-C., et al.: APAMAT: a prescription algebra for medication authoring tool. In: IEEE Conference on Systems, Man and Cybernetics (2006)