Universitetet i Oslo
Institutt for informatikk

Platform Independent User Interface Development for Mobile Systems

Petter Aamot Vangstein

Cand Scient Thesis

October 2004

Abstract

An important trend today is the demand for increased flexibility of where and how work is done. The physical work environments become more diverse and the border between work and leisure decrease. This demands flexibility of how, where, in which situations, and from what types of devices applications and services can be accessed. It is assumed that there will be a need for richer and more dynamic user interfaces on new information processing devices than are possible with HTML/XML. As a consequence of these trends the users will choose mobile devices for their needs. These devices will probably have different specifications such as type, operating system, user interface styles and design. This will increase the need for platform independent user interface development (multi interfaces). The goal of this work is to explore existing task- and model-based user interface development approaches, and then see how they can be implemented towards mobile systems supporting different types of devices and user interface types. We propose a framework for multi-interface user interface development for mobile systems. It is based on existing task- and model-based approaches and by this keeps both a user centered and designer centered approach. Model-based theories are also used to solve the demand for platform independence. This is done by creating patterns in such an abstraction level that the platform specific characteristics are avoided as elements in the language for dialogue modeling. These models are then, with a limited set of platform specific parameters, realized to different types of platforms thru code generation. The framework is used on a comprehensive case-study with a following evaluation and discussion. The case study showed that the framework solved platform independency, but still need some more work on the set of patterns and their abstraction level. Despite this we argue that it is a promising approach and a good base for further work on the topic.

Acknowledgements

This Master Thesis is submitted in partial fulfillment of the Cand. Scient Degree in Informatics at the Department of Informatics, University of Oslo (UIO). The thesis work was conducted at SINTEF ICT. I wish to thank my supervisor John Krogstie for his guidance and patience with me. Erik G. Nilsson, Hallvard Trætteberg, Tomas Norheim Alme and Ben Kirk receive thanks for useful and interesting discussions. Special thanks to my wife, Hege, who gave me the final motivation to finish the thesis.

Table of Contents

Table of Figures

1. Introduction

In this chapter we state the purpose of the thesis. A brief introduction to the problem domain is provided and a statement of the primary goals is presented, before finishing with an overview of the rest of the thesis.

1.1 The Domain of Interest

An important trend today is the demand for increased flexibility of where and how work is done. The physical work environments are getting more diverse and the border between work and leisure decrease. This demands flexibility of how, where, in which situations, and from what types of devices applications and services can be accessed. It is assumed that there will be a need for richer and more dynamic user interfaces that are possible with HTML/XML. As consequence of these trends the users will choose mobile devices for their needs. These devices will probably have different specifications such as type, operating system, user interface styles and design. This will increase the need for platform independent user interface development (multi-interfaces) [1].

There is a plethora of different types of mobile devices, but broadly what that distinguishes them are the output- and –input characteristics.

The two main categories of output types are screens and speech/sound. Speech and sound based systems are probably the most complex systems to evolve, but the great advantage is that there are no differences between different types of devices of how they should be evolved.

A challenge in evolving systems for screen based devices is the wide range of different types of screen sizes. This spans from screens that only fit a few characters like clocks to large “wall size” screens. The three most common categories of input characteristics are keyboard-, pen- and speech-based.

There are different sub-categories within keyboard-based devices. These sub-categories span from mobile phone keyboards, mini size QWERTY keyboards and full size QWERTY keyboards.

Pen based systems also have different sub-categories in how complex the pen features are. Some systems only support mouse features while others support full hand writing recognition.

The challenge is to find good development methods for evolving user interfaces that function on many different types of platforms. Nilsson [1] evaluates some of the available approaches based on this need:

1. Develop clients from the bottom for each platform. The advantage of this approach is that there is no demand for special tools and languages, but the disadvantage is that it is very time consuming and also hard to maintain.
2. Platform independent user interface libraries. An example of this is Java, where the same code can run on different types of platforms that support it. The disadvantage is that these libraries often only support one type of user interface. If the system should support different types such as graphical and speech based user interfaces, different libraries must be used for each type.
3. Generic clients. Web-browsers are an example in which user interfaces are specified in markup languages as HTML, XML, and WML etc. The advantages are that it easily functions on different types of platforms and the technology has also proved to be very scalable. The disadvantages are that there are limited functional possibilities, it is complicated to maintain transactions and different types of devices support different types of protocols (mobile phones often only support WML).

4. User interface modeling. This approach is based on specifying user interfaces as models in a conceptual modeling language. The advantage of this approach is that a model can, in theory, be made to suit many platforms. Nilsson states that model based user interface modeling for mobile systems has good potentials, but that it is not fully exploited by any language or tool yet. Nilsson has identified three problems with the current languages and tools:

    - They are generally not on a high enough conceptual level. The tools usually consist of simple elements as 'buttons', 'text fields', 'menus', 'radio buttons' etc. These can be used to put together concepts on different abstraction levels: E.g. 'radio groups', which are groups of 'radio buttons' or 'choice elements' which are on higher conceptual level and can consist of 'radio buttons', 'drop-down lists' or 'list boxes'. This concept makes user interfaces into instance hierarchies of these elements. This works fine as long as all platforms supports these hierarchies, but these hierarchies are often tied to screen sizes so it will not support platforms with different screen sizes such as both a mobile phone, a PDA and a PC. In these cases separate instantiation hierarchies must be made for each platform. A solution to this issue may be dividing the model to be platform independent and, for each supported platform, a platform specific model. But the problem with this solution is that the platform specific models often turn out large and complex because there are too many differences between the platforms.
    - Model generated user interfaces are often less user friendly. This because the tools do not often exploit the possibilities and qualities for each platform. A solution to this is platform specific mapping rules, or manually changing generated code for each platform. The disadvantage with this solution is that it is very time consuming and complicated.
    - It is often restrictive on the type of user interface that can be specified. A common restriction is to use form-base database applications.

Nilsson concludes that all these methods have many disadvantages, but that user interface modeling has potential that is not yet fully utilized and may be a good starting point for a new approach.

[2] states that existing approaches for supporting usability is well suited for developing stationary and in-office systems, but does not cover the necessary span of contexts where mobile devices are used. This is critical because users of mobile systems are characterized by frequent context changes. Mobile systems should therefore, as a consequence, adapt the context changes when it is relevant.

As a result of the complex issues in mobile infrastructure and usability (i.e. frequent context changes), development processes with structural qualities open for flexibility and creativity is a demand. A model driven process can be such an approach. [2] presents the following qualities of model driven development processes:

- Explicit representation of goals, organization and roles, persons and knowledge, and processes and systems.
- An effective tool for communication and analysis.
- A foundation for design and implementation, through code generation, generating calls to already specified components or as documentation.
- Available documentation as a foundation for extensions and personalization.

1.2 Goal

The goal of this work is to explore existing task- and model-based user interface development approaches, and then see if and how they can be implemented towards mobile systems supporting different types of devices and user interface types. We will propose a framework containing a development process and a methodology based on this analysis. Important overall criterias will be:

- Support user centered development reflecting the goals, organization and roles, persons and knowledge, and processes and systems.
- Support the designers by: Tools for communication, analysis and documentation. A foundation for design and implementation.
- An appropriate abstraction level that does not constrain the compatibility towards different mobile platforms.
- Make sure that the approach does not affect the user friendliness and use the possibilities and qualities for each supported platform.

The result will be tested in a defined case study.

1.3 Thesis Structure

The following describes the structure of this thesis:

Chapter 2 Background: Introduces background information on concepts and technologies used in the thesis.

Chapter 3 User Interface Modeling Using Abstract Dialogue Patterns: Is a detailed description of the proposed approach.

Chapter 4 Case Study: Introduses a potential field and system where we use the framework for test and evaluation.

Chapter 5 Evaluation and Discussion: This chapter evaluates the proposed framework. Discussion and results are presented.

Chapter 6 Conclusion and Further Work: The fulfillments are evaluated and discussion is concluded, contributions are presented and further work proposed.

Appendix A: A short introduction to the task model language TaskMODL.

Appendix B: A short introduction to the dialogue model language DiaMODL.

Appendix C: Description of the code generator tool ‘UmlAPI’, which is used in the case study to generate code from the platform specific UML models.

Appendix D: Include source code for the components in the framework, and the case study.

2. Background

The main aim of this chapter is to give the reader insight of a task- and model-based user interface design, presenting current state of the art in the field.

Both task and model-based design of interactive systems focus on the use of integrated modeling notations to support design at various levels of abstractions. These models are expressed using formal and/or semi-formal notations and they may be defined relations between the different models. Both approaches have supporting tools that are aimed to automate the design activities or to support designers in their work, but they are distinguished by their ability to express aspects of usage of interactive systems. This is because model-based approaches tend to not model how a system might be used to accomplish the user's tasks. Model-based approaches also tend to have greater interest in design support tools while task-based approaches focus on the design process [3].

Model-based approaches give the designer better support in constructing user interfaces. This is done by letting the designer express the user interface in a high level of abstraction, focusing on the behavior. These models are then used by automated tools to generate the executable interface. These approaches say nothing about how the abstract models should be constructed and limit their interest to those processes that occur in the transition between abstract and concrete design solutions and therefore they are not user-centered [3].

Task-based approaches have their primary interest on processing design solutions from information about the users' tasks. This is to make sure the system is compatible with the task it's meant to support. The tool developed for this approaches has mainly been model editor tools or low level generator tools like those developed for model-based approaches. [3]

2.1 Task Modeling

Developing information systems demands an analysis of the nature and structure of the work to be done, including users and the environment related to the work [3-5]. This is because the nature of information systems are information entities working together to achieve goals [4].

From this we can extract the main elements of task modeling:

1. **Task**; an activity that can have several goals. Work is described by tasks [4, 5]. [5] also present a view on tasks where there is a one-to-one relation between task and goal. In this definition a goal can not be reached by different tasks. We will use the first definition. A task can be decomposed into sub-tasks and also be categorized into different complexity levels: A 'unit-task' represents the lowest level that people can relate to work and are often role related[1][4]. A 'basic-task' represents the lowest level that a tool will use to perform a task[12][4]. Tasks are often organized in hierarchies where tasks can ***trigger*** an ***event*** that certain tasks respond to. This makes it important to understand the flow of tasks. [4, 5]
2. **Agent**; represent the users that can be humans or another systems. It indicates classes of individuals with certain characteristics.[4, 5]
3. **Role**; is the relationship between agents and tasks. An agent can have several roles. A role has a collection of tasks it is responsible for. [4, 5]
4. **Situation**; is the environment/context where the task is performed. The environment can be physical, conceptual or social and includes objects relevant to the task. [4, 5]

The process of task modeling consists of two steps: Model the existing as-is situation to help the designer understand the current users, the users' existing tasks and the results. The second step is to envision the to-be model describing the users and the tasks that will be influenced by the new system. [3, 4]

2.1.1 As-Is Task Model

The main task of modeling the as-is model is gathering information about the existing situation. [3] presents the following steps in this process:

- Identify characteristics of specific tasks.
- Analyze many different users performing each task and identify all variations and individual differences.
- Produce a task description for each user on each task.

[3] names this information the Specific Task Model.

There are different techniques supporting gathering of information to a specific task model, for example observations, interviews and questionnaires. [3] gives the following heuristics for choosing techniques:

1. "Always use more than one data collection technique since any technique will only give partial information about a task."
2. "Direct and indirect observation techniques are well suited for identifying patterns of behavior, temporal aspects of tasks, behaviors and procedural aspects of tasks, but are poorly suited to predominantly cognitive tasks. The analyst needs to be aware that observations are time consuming, cannot be used in isolation, and that interpretation of observations can involve a degree of inference on the part of the analyst."
3. "Interviews provide a useful technique for identifying general rules, background knowledge, conditions and constraints upon tasks, the goal structure of a task and dependencies between tasks, but are poor at identifying temporal and procedural aspects of tasks. The analyst should be aware that people are better at remembering conditions given actions, rather than actions given conditions."
4. "Questionnaires are best used to obtain shallow descriptions of task properties, and are useful to identify objects and attributes of a domain and their structural (class and component) properties, but are poor at providing detailed task information or information about context sensitive task behaviors."

A general rule is that the analysis should focus on identifying all variations, individual differences for each task, and general characteristics among them. The analyst should also identify objects and their attributes which are relevant to the tasks[3].

This information is used to produce a composite task model emphasizing the goals and the tasks that can be performed to achieve the goals. The tasks and goals are also decomposed into sub-tasks and sub-goals including the flow between them like compulsory or optional tasks and alternative routes to achieve the goals. The task tree can also be enhanced with time relations between tasks. [3] names this a Composite Task Model and presents the following steps to construct the artifact:

- From each task description for the same goal, produce a composite task model which includes all the different ways of achieving the goal.
- Identify all the different ways of achieving the same goal.

- Resolve conflicting descriptions (e.g. where the same course of action appears to lead to two or more different goals.)
- Identify optional aspects of a task (i.e. where there is a high degree of variance and a low occurrence across the specific task models.)
- Identify compulsory aspects of a task (i.e. where there is a low degree of variance and a high occurrence across the specific task models.)
- Identify commonalities of behavior, patterns of behavior and common objects across the different tasks
- Identify constraints and dependencies across tasks.
- Identify the different objects and typical instances of objects where there are a number of different examples of the same object across the different tasks.

Identifying objects is strongly related to the design of the data structure in the final design and implementation. These are objects related to the tasks and can be physically or mentally present. Information systems focus on *information passing* and a task's objects typically have an initial and final state in terms of object attribute values. [5]

User Context

The users of mobile systems are characterized by ferequent changing context [2] and we will claim that an extensive modeling of user context (situation) is critical. User context can be organized into the following categories [2] (Figure 1):

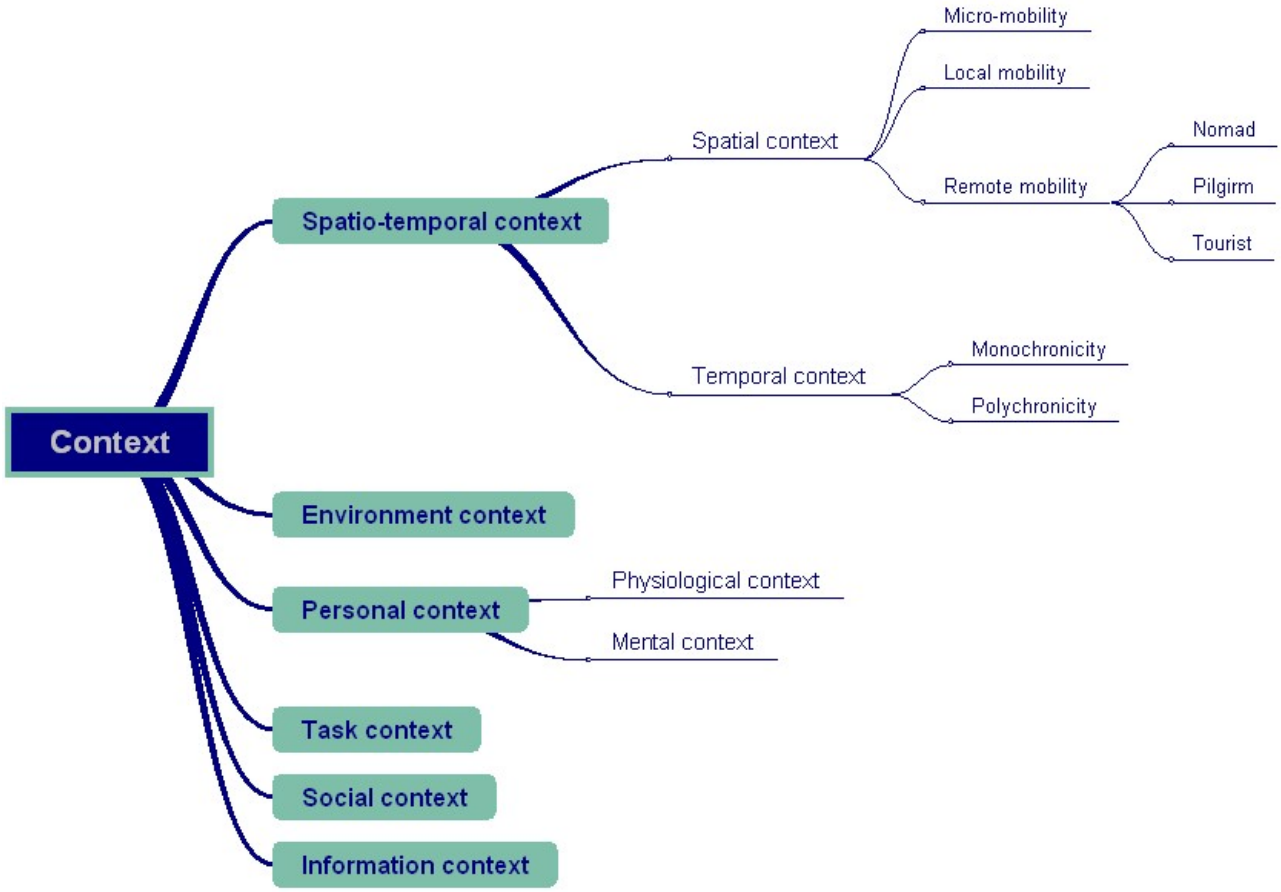

Figure 1: User Context Categories

1. Spatial-temporal context. Describes aspects related to time and space, and contains attributes as time, location, direction, speed, patch, and space.
2. Environment context. Describes entities surrounding the user, e.g. physical objects, available services, temperature, light, humidity and noise.
3. Personal context. Describes the condition of the user. This category can be divided into to sub categories:
    a. Physiological context. Contains attributes as pulse, blood pressure, weight.
    b. Mental context. Describes user conditions as mood, anger, stress and knowledge.
4. Task context. Describes the task the user will accomplish or the goals.
5. Social context. Describes social aspects as information of friends, family, and colleagues. This also describes which role(s) the user possesses in different situations.
6. Information context. Describes available information.

Spatial-temporal context

Spatial-temporal context includes the user entities time (temporal) and space (spatial).

Spatial Context

[6] defines space to be the structure of the world as a three dimensional environment including objects and events with relative positions and directions. An important quality is that the spatial attributes are global and persistent. The following can be described by analyzing a user's spatial environments:

- Proximity and events between objects. The degree of direct interaction between people and objects decrease when the distance between people and objects increase. This can be helpful when relating people to activities and other people.
- Partitioning of space from the conception of proximity. This can be used to understand the degree of interaction between people and objects.
- Presence and consciousness to other objects and events, which can be used when structuring our own actions in relation to present persons and activities.

[7] introduces the following degrees/types of spatial mobility:

- Micro-mobility: Mobility within a small area. The example Luff and Heath[7] refers to is a consultation between a doctor and a patient. In this situation it can be of great importance that the doctor can be able to move the patient record from the desk to his knee to be able to examine the patient while making notes and all the time focus on the patient.
- Local mobility: Real time interaction between people and technology in same location.
- Remote mobility: Describes synchronic and asynchorinc collaboration between people that move around in different locations.

There are a number of ways to divide remote mobility into sub categories. One way is nomad, pilgrim and tourist. [8] A nomad user moves frequently and in unpredictable patterns. A pilgrim user moves frequently, but in predictable patterns. A tourist user moves round in predictable and planned patterns. Nomads and pilgrims are business users while tourists are recreational users.

Temporal mobility

Monochronicity and polychronicity are concepts used to describe organization of time (temporal mobility). Monochronicity is sequential organization of time, and polychronicity is the opposite where the users work on many tasks simultaneously. [2]

Most mobile devices have been limited to a screen that is best suited for monochronicity. A common trend today is that enterprises demand processes to follow strict sequential patters. Therefore, users are often in a paradox to contexts that demand polychronicity. Examples are consultants that often work on many projects simultaneously, or users that wish to use the same tool, e.g. diaries, both for private and professional use. [2]

Social context

In many cases analyzing spatial context will not be enough to build a system, because it only includes physical aspects and excludes the social aspects[6]. [9] states that the social context the building is going to be used in is more important than the physical environment in construction and building architecture. Social context describes the social aspects of the environments the users are in, like roles, social rules, and guidelines. Social context is what makes a place more than just a physical coordinate in space. A physical place is most often a location in space, but has also a set of social rules and guidelines. Space is persistent, while place is transient from how the location in space is used.

An important factor in social context is that it does not need to be related to a physical location. An example is MUD (Multi-user dungeon). Social context can in some cases be tied to virtual environments that simulate physical locations.

Agre [10] introduces institutions as an important element to describe and analyze social context. This theory is based on relations between architecture, institutions and people. All institutions have a set of social roles often in hierarchies, and rules and guidelines. Those rules and guidelines may be unwritten. Agre uses a theater as an example of such an institution. A theater has the social roles public, artists, ticket officers etc. The public's rules and guidelines may be that mobile phones should be set to silent mode while the play is on. An important aspect is that the phone itself does not know these social roles, and it relies on that human beings to configure the phone to suit the social roles. Hospitals are another example. The social roles are doctors, nurses, patients, and it-professionals. The roles are hierarchical with guidelines and responsibilities.

[10] introduce a framework for analyzing this phenomenon. The framework is divided in three levels:

- Architecture – include attributes that describes physical structures in the environment such as buildings, windows, doors where the surrounding the user.
- Practices – describes concrete routines developed in a social environment. This is the same as task context described earlier.
- Institutions – is persistent structures of human relations created by social roles and rules.

[10] states that these three levels historically are closely related. For example, architecture and institutions are related because almost every building is designed to suit an institution. Hospitals are designed to suit doctors, nurses, patients etc., where the different roles have their own rooms etc. These two levels are often persistent. New technologies misrepresent this picture because they break the relations between architecture and institutions by making the institution's activities independent of the building.

We will claim that Agre's institution theory can ease the analysis of social context because in most contexts we can assume that social context is defined in institutions and that we can model the user's social context as a relation to its defined role in the institution's hierarchy. These relations will change when the users move to other institutions. A possible approach for institutions is to define institutions-patterns.

2.1.2 To-Be Task Model

The existing task model, requirements like constraints on the possible design solution and the overall problem statement of the existing situation are then envisioned into a to-be task model. Requirements can for example include integration with existing information systems, security policies etc.[3] An example of an overall problem statement can be: Increase the efficiency and decrease errors and by this increase the profitability.

The first step of the process is to identify the scope of the new system. This is done by identifying any tasks that should not be handled by the system, tasks that can be avoided, tasks that can be fully handled by the system, tasks that can be fully handled by the user and tasks that are handled by the user interacting with the new system.[3]

The next step in the process is then to analyze the tasks within the scope and seeking to improve them. The steps in this process are: [3]

- Identify where sequences of activity can be made easier to perform, e.g. by removing unnecessary constraints between activities, making it possible to interleave activities and/or carry out activities in parallel.
- Create more powerful objects by composing and combining individual objects, making it possible to carry out actions on those composed objects.
- Bring together information that is distributed across several objects but all required at the same point in the task.
- Ensure that the requirements are supported, like security procedures.

2.2 Dialogue Modeling

We distinguish between two perspectives in dialogue models: Abstract dialogue models that focus on dataflow, activation and sequencing. Concrete dialogue models are concerned with the characteristics of the supported device and the dialogues relation to this. This includes for example windows, speech, pen and mice[11]. The latter is envisioned from the abstract models.

2.2.1 Abstract Dialogue Modeling

In the abstract dialogue model task models are decomposed further from sub-tasks into dialogue elements. These dialogue elements should be on such an abstraction level so they will fit any user interface platform.

The main concern of the abstract dialogue modeling is task decomposition into dialogue elements. [3] presents the following heuristics for this process ([3] do not distinguish between abstract and concrete dialogue modeling, so these heuristics are modified to meet this approach):

- Reflect the goal, sub-goal and action decomposition in the overall structure of the interface.
- Group interface components that support closely related parts of the task.
- Let the lowest level of task decomposition (i.e. the actions) be the strongest determinant of task structure.
- Group interface components by placing them in close spatial proximity, or in close temporal proximity in the dialogue structure.
- Use task actions and objects to determine the components that will actually appear in the interface and the ways in which those components can be manipulated.
- Use actions to suggest commands.
- Use objects to suggest information to be manipulated and/or displayed.
- Use action-object groupings to indicate information that can be manipulated in particular ways.

- Let sequencing information in the task model be the major determinant of the dialogue structure of the interactive system.
- Do not violate task sequencing in the interface design.
- If desirable, relax sequencing constraints in situations where safety conditions will not be violated.

The problem with platform independent user interface development for mobile devices is that they often do not utilize all the qualities and possibilities on each of the supported platforms. This is in terms of poor user friendliness. A common solution to this problem is dividing the models into one platform independent model and a platform specific model for each supported platform, but often the similarities are few and it results in large complex platform specific models. The main challenge for a modeling language is therefore to keep an abstraction level that is generic enough to cover different interaction mechanisms on different platforms while still maintaining its ability to function as a meaningful specification for the concrete platforms [12].

Nilsson introduces a theory that uses pattern based compound user interfaces as elements in modeling languages. The core of the method is pattern modeling, where a pattern is a compound of user interface components. A component can also be a compound of other components. A critical factor is that the user interface component's abstraction level has to be high enough to function on many different platforms. All the patterns will need transformation rules for each supported platform to make code generation possible. One pattern can also have a set of transformation rules based on user preferences, style, user context etc.[12]

As a consequence, a user interface model that illustrates a pattern will have to consist of the pattern instance, supported implementation platforms and which of the available transformation rules should be used for each platform.

As an example Nilsson uses the conceptual pattern “Composite” from the book “Design Patterns” of “The Gang of Four” [13]. This pattern can be used to describe objects that can be of two types: compound objects or leaf objects. Compound objects can have many child objects that can either be a compound object or leaf object. (Figure 2)

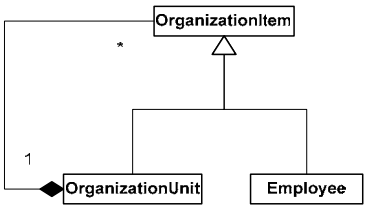

Figure 2: Composite Pattern

Common examples of this pattern are the file explorer in the "Windows" operation system, when it's organized in a tree structure, and a tree representation of an organization. The two types of objects in this example are departments and employees. The goal is to navigate to an employee to read registered information. This conceptual pattern can be represented in different ways on different platforms without the goal changes. (Figure 3) and (Figure 4) shows examples of how the composite pattern can be transformed between a stationary PC and a Tablet [12].

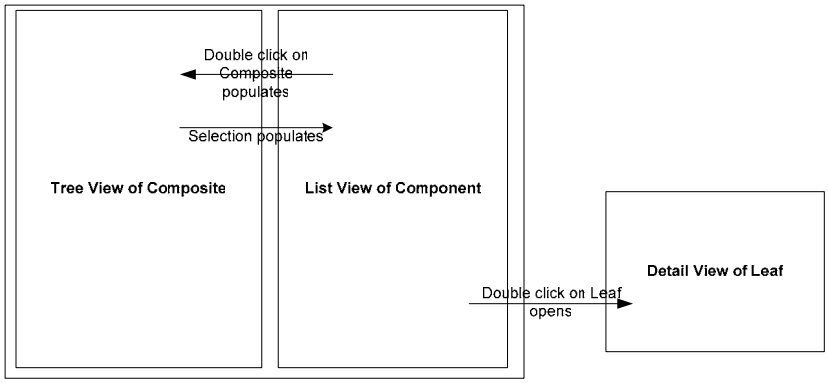

Figure 3: Composite Pattern PC/Tablet Example 1

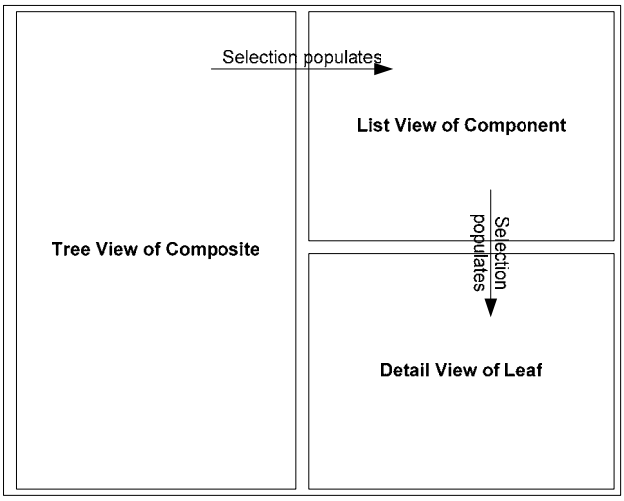

Figure 4: Composite Pattern PC/Tablet Example 2.

(Figure 5) and (Figure 6) are suggestions of how this pattern can be represented on a PDA[12]:

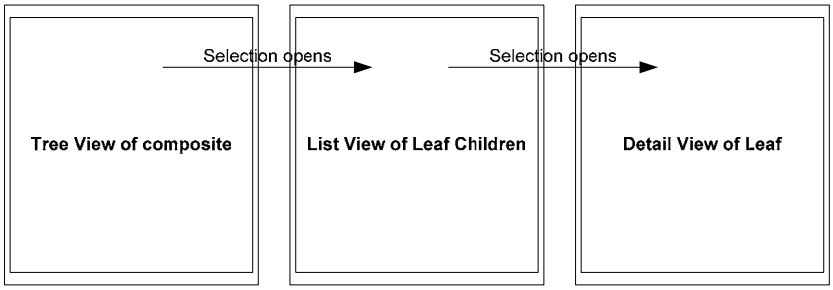

Figure 5: Composite Pattern PDA Example 1

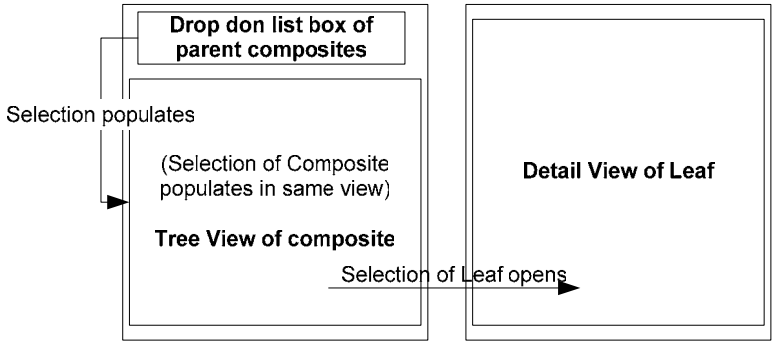

Figure 6: Composite Pattern PDA Example 2.

In a speech based system the transformation rules can be that the system first reads the objects in the first level and indicates if they are compound or a leaf objects. The user can then select one of the objects. If the selected object is a leaf object the system will read out all its attributes or if it is a compound the system will read out all its child objects.

These examples state that the abstraction level does not make obstacles for transformation to different platforms, but it's also important that the abstraction level is not too high because then the transformation rules will be too complex and extensive.

The transformation rules also have to include information of how selection and double clicking elements function, as well as drag-and-drop functionality etc. Instantiations of model patterns also include possible column names, sorting possibilities, data type presentation, trigged actions etc. This should, if possible, only be a part of the pattern and not be included in the transformation rules [12].

Nilsson states that the critical factor for this theory to be realized is that a set of pattern-based user interface components that cover the most essential requirements are developed with transformation rules for a set of relevant platforms. He assumes that with a set of good conceptual patterns the number of patterns will be limited. He also assumes that each system will only support a limited number of platforms. In this way there will be a limited number of patterns and transformation rules and one will avoid large complex platform specific models.

2.2.2 Concrete Dialogue Modeling

The concrete dialogue models are evolved from the abstract dialogue model and are platform specific models that are tailored to each relevant platform. Concrete dialogue models consist of concrete dialogue objects that implement the advantages of the current platform and are related to abstract dialogue objects which they are mapped from.

2.3 Summary

We have described task- and model-based development. Task-based approaches have their primary interest on processing design solutions from information about the users' tasks to make sure the system is compatible with the task it is meant to support. Mobile workers are often characterized by frequent context changes which mobile systems should tailor and adapt to. We gave therefore an introduction to the user context concept. Model-based approaches give the designer better support in the construction of user interfaces by letting the designer express the user interface on a high level of abstraction, focusing on the behavior. We have also introduced Nilsson's [12] theories that uses the model-based approach to create multi-interface mobile systems. In the forthcoming chapter we will propose a framework based on these theories.

3. User Interface Modeling Using Abstract Dialogue Patterns

In this chapter we propose a framework for developing platform independent user interfaces for mobile systems. The approach is a combination of the methods described in chapter 2.

3.1 Task Modeling

Our approach will try to combine task- and model-based user interface design to profit from both approaches' qualities. From the previously described process of task modeling, that which distinguishes context modeling of in-office systems and mobile systems is the lack of analyzing user contexts. For this approach analyzing user context will be important to find appropriate user interface types and adequate device types. [12] states that limiting the number of supported platforms is important to realize the approach using compound user interface components. We suggest that the task models are extended with the following context specifications:

- Identifications of any spatial characteristics that can affect choice of user interface type or type of device. This should be described by using the sub-categories of spatial context.
- Analyzis of the task flow (temporal context like monochronicity and polychronicity) in the context of mobility. For example it is harder to handle polychronicity on small screen devices and strict speech based user interfaces.
- Identifications of any environmental characteristics that can affect the user interface or type of device.
    - Noise, light that will give hints to user interface types.
    - Any transportation objects like vehicles. For example adequate device size can be affected if all tasks are performed from within a vehicle.
    - Other devices or systems that are present that can be used. For example, a stationary screen present in a sub-task can be used in a multi channel manner to extend the qualities of the main device.
- Identify any social restrictions like noise or light that will affect the choice of user interface type or device.

The first four are relevant to the to-be task model while the last one should be modeled in the as-is task model. Based on this information a list of suited devices and user interface types should be included in the to-be task model.

In our point of view, user contexts are inherited by sub tasks and are only specified in a sub-task if there are changes.

We use [11]'s TaskMODL as our modeling language. Appendix A includes a short introduction to the language. There is no specialized modeling tool available supporting this language so we use Microsoft Visio [14] to do the modeling.

3.2 Dialogue Modeling

Our approach has relations to Object Management Groups' (OMG) Model Driven Architecture (MDA) approach. They meet the platform portability by distinguishing between platform independent models and platform specific models[15]. In our approach the abstract dialogue model works as a platform independent model and the concrete dialogue model as platform specific model. We use the principles presented in [12] to avoid the problem of too extensive platform specific models.

3.2.1 Abstract Dialogue Modeling

We have created a small set of abstract platform independent patterns that we think will cover the most common needs for a computerized information system, as elements in our dialogue models. This among others is based on abstract interactor controls in [11], theories in [16] covering automatic user interface generation from data models, and by analyzing existing computerized information systems and their behaviors. Our abstraction level is a bit lower than intended in [12], but there is no obstruction to use patterns in a higher abstraction level in the framework. We use [11]'s DiaMODL (appendix B includes a short introduction to the language) for modeling dialogue models and each pattern is modeled as a DiaMODL interactor marked with the pattern's inherent stereotype. The stereotype element is not in the specification of DiaMODL, but DiaMODL use elements from UML so we have introduced it in DiaMODL to decrease the complexity of the transformation rules and relations between the models. Each pattern does also have a class specification for an abstract class that all the platform specific classes should inherit. In the case study, introduced later in this thesis, we use Java as the implementation language so therefore we have implemented these classes in Java. These implementation support J2ME [17]as well as J2SE [18].

The specifications of the abstract patterns are presented in the forthcoming sections:

Element Selection

The interaction control presents a list of domain objects of the same type and lets the user select one from the set. As input the control takes a list of elements and one of the elements may be marked as a default selection (Figure 7). The name of the inherent stereotype is '*elementselection*'.

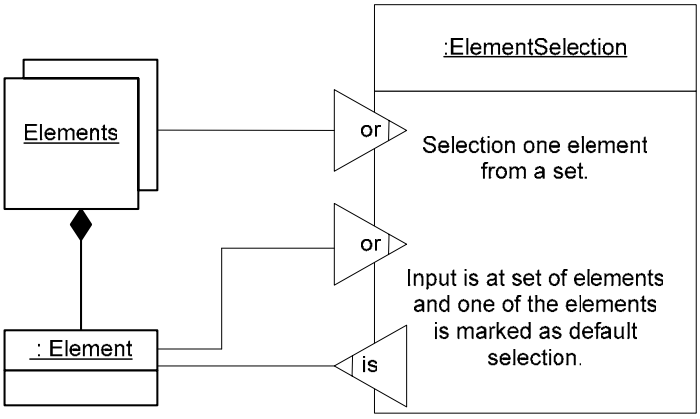

Figure 7: Element Selection

The abstract class for the element selection pattern covers setting the element set using the method ‘public setElements(elements : Set) : Void’, setting default selection using the method ‘public setSelectedElement(element : OclAny) : Void’, setting callback listeners using the method ‘public setListener(listener : jinteractioncontrols.IActionListener) : Void’ (IActionListener is a interface that callback subscriber must implement (Figure 8)) and handling callbacks when selections are performed. The method ‘public getSelectedElement() : OclAny’ can be used to fetch the selected element after a callback is performed. (Figure 9) The Java implementation of this class is presented in Appendix D, page 92.

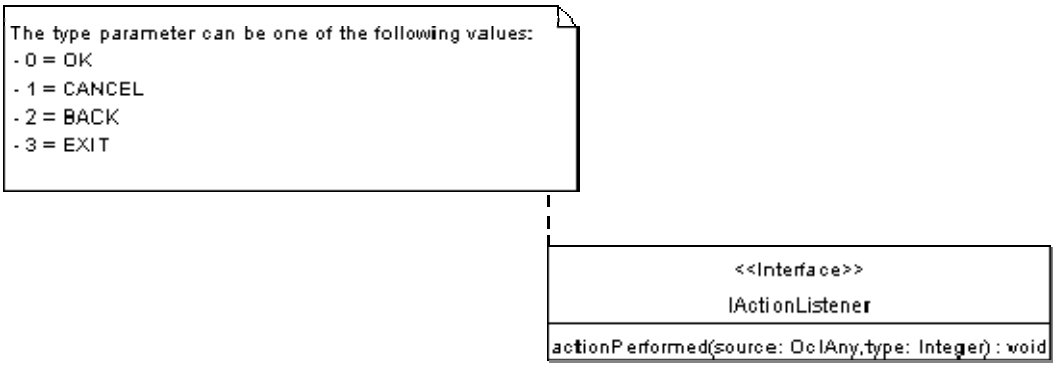

Figure 8: IActionListener Interface

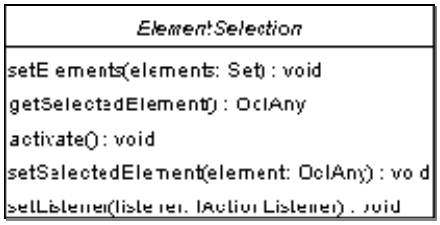

Figure 9: ElementSelection Abstract Class

Subset Selection

Presents a set of domain objects of the same type and lets the user select a subset from the original set. As input the control takes a list of elements and a subset of the elements may be marked as default selection. (Figure 10) The name of the inherent stereotype is ‘*subsetselection*’.

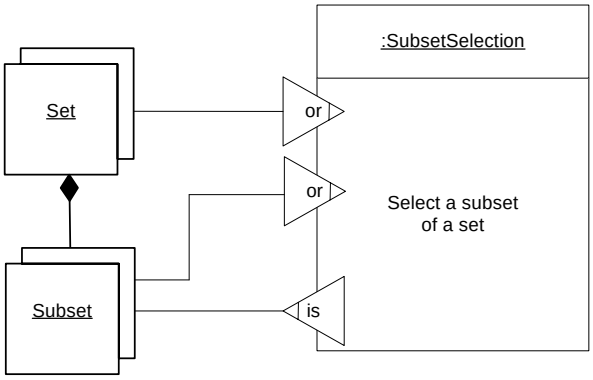

Figure 10: Subset Selection

The abstract class for the subset selection pattern covers setting the element set using the method 'public setElements(elements : Set) : Void', setting default selection using the method 'public setSelectedElements(elements : Set) : Void', setting callback listeners using the method 'public setListener(listener : jinteractioncontrols.IActionListener) : Void' and handling callbacks when selections are performed. The method 'public getSelectedElements() : Set' can be used the fetch the selected elements after a callback is performed. (Figure 11) The Java implementation of this class is presented in Appendix D, page 94.

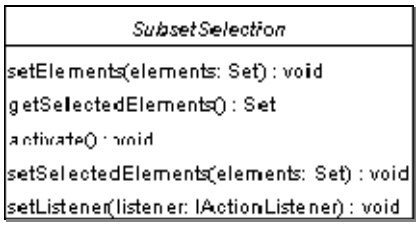

Figure 11: Subset Selection Abstract Class

Hierarchical Selection

This control presents a hierarchy of composite and leaf objects and lets the user traverse the hierarchy and select a leaf object. It takes a composite element as input. (Figure 12) The name of the inherent stereotype is '*hierarchicalselection*'.

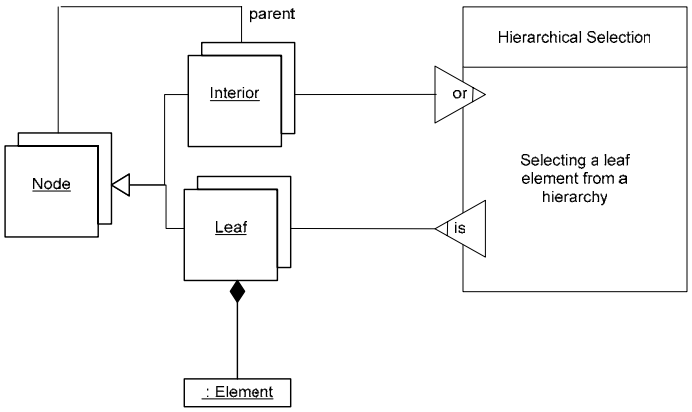

Figure 12: Hierarchical Selection

The abstract class for the hierarchical selection pattern covers setting the root element using the method ‘public setRootElement(rootElement : OclAny) : Void’, setting callback listeners using the method ‘public setListener(listener : jinteractioncontrols.IActionListener) : Void’ and handling callbacks when selections are performed. The method ‘public getSelectedElement() : OclAny’ can be used to fetch the selected element after a callback is performed (Figure 13). The Java implementation of this class is presented in Appendix D, page 95.

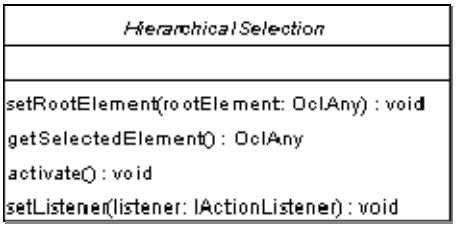

Figure 13: Hierarchical Selection Abstract Class

Edit/View Element

The interactor presents an entity for the user. The entity attributes may be editable, but this is specified in the concrete model. It takes an entity object as input (Figure 14). The name of the inherent stereotype is ‘*elementview*’.

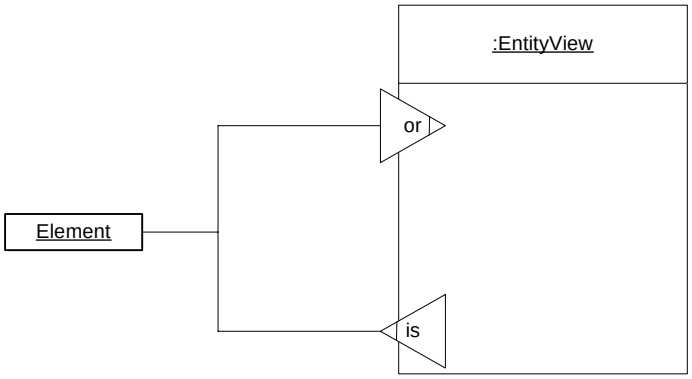

Figure 14: Edit/View Element

The abstract class for the hierarchical selection pattern covers setting callback listeners using the method 'public void setListener(jinteractioncontrols.IActionListener listener)'and handling callbacks when selections are performed (Figure 15). The Java implementation of this class is presented in Appendix D, page 99.

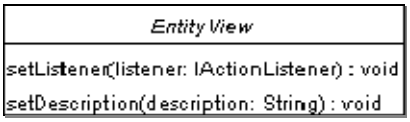

Figure 15: Edit/View Element Abstract Class

Interactor Selection

The interactor presents a set of available interactors and lets the user select one. The interactor will then make the selected view available. It takes a set of interactor references as inputs (Figure 16). The name of the inherent stereotype is '*interactorselection*'.

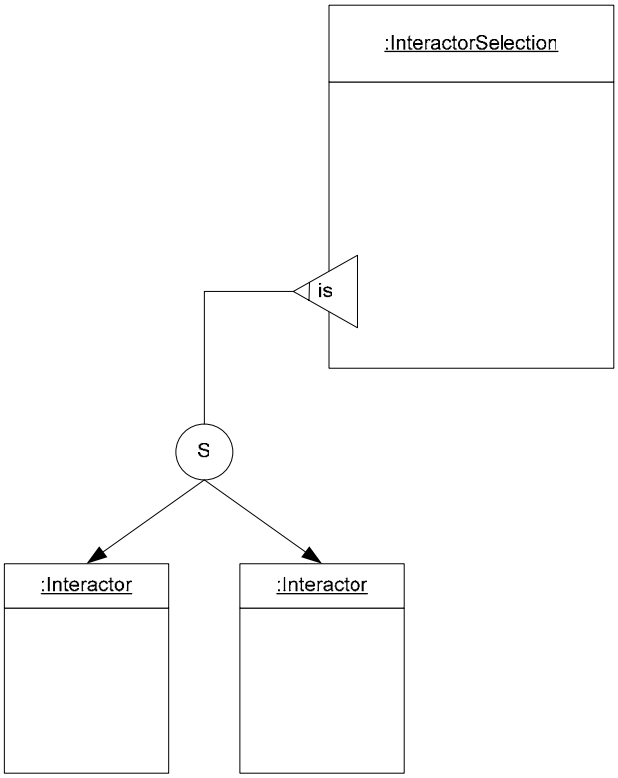

Figure 16: Interactor Selection

The abstract class for the element selection pattern covers adding command elements using the method 'public addCommand(jinteractorcontrols.Command : Set) : Void', setting callback listeners using the method 'public setListener(listener : jinteractioncontrols.IActionListener) : Void' and handling callbacks when selections are performed. The method 'public getSelectedCommand() : Command' can be used to fetch the selected element after a callback is performed (Figure 17). The Java implementation of this class is presented in Appendix D, page 97.

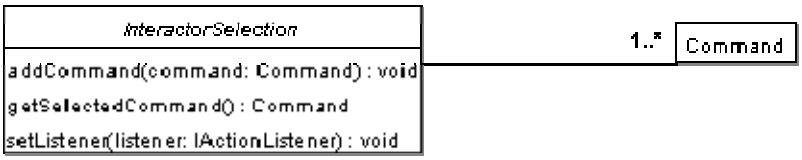

Figure 17: Interactor Selection Abstract Class

3.2.2 Concrete Dialogue Modeling

Each of the abstract patterns needs some platform specific information to be realizable. This for example includes which attributes in a data object should represent the object in an element selection interactor object or what the descriptive label on an edit/view entity attribute should be. The case study will later demonstrate this. The platform specific specifications are modeled as OCL [19] expressions. Each abstract pattern does also need transformations rules for each relevant platform. Together the platform independent model, the platform specific model and the transformation rules can be transformed to code tailored to each platform. Initially we experimented with platform specific user interface components directly supporting the platform independent patterns to avoid code generation. These components took the platform specific specifications (OCL expressions) as parameters, but these techniques were based on reflection to handle the metadata and did not work because of J2ME's lack of support for reflection. We therefore changed to a strategy that generates classes tailored to the specific platform and including the OCL expressions represented as java code. The generated class inherits the existing platform independent abstract classes. The implementation of the generated platform specific interactor is based on characteristics of the relevant user interface types and devices. Another important factor is already existing user interface components in the chosen development platform. The latter is probably most relevant for graphical user interfaces where there exists a wide range of user interface components. Speech development platforms for example seem to not include components on a higher abstraction level other than text-to-speech and speech-to-text. Implementation of these controls will be covered in the case study.

We have also created platform specific abstract classes in Java for each pattern to make the transformation rules less complex. These include all the pattern's platform specific commonalities and inherit the platform independent abstract class inherent to the pattern. The generated class will then only include the platform specific specifications and inherit the abstract platform specific class.

There is no specialized modeling tool available supporting DiaMODL so we use Microsoft Visio [14] and as a consequence of this we do not have the ability to export the models in an appropriate format that a transformation tool can handle. We have created a "work around" to solve this issue. In our approach the concrete models have references to each interactor object in the abstract dialogue model by UML classes [19] specified for each platform with the same stereotype as the interactor object. There exist tools to directly export such models in appropriate formats such as XMI [20]. We use ArgoUML v0.14 [21]. This terminates the possibility to use the dynamic characteristic in dialogue languages, and we can therefore not automatically generate flow and layout between the interactors. Despite this shortcoming we believe that we can demonstrate the ideas of the approach and that it still will be a useful framework. The flow and layout code between the interactors is therefore not automatically generated in the current solution.

We have also created a tool to be able to automate generation of user interfaces for different platforms: 'UmlAPI' (see appendix C) is a code generator that takes XMI [20] and templates specifying the transformation rules as input. There are other tools like 'UmlAPI', but they seem to only cover a part of the UML metamodel. E.g. UMT [22] does not support tagged values and OCL constraints which is a used in our approach.

3.3 Summary

We have here described the fundamentals of the proposed framework: Which existing theories it is based on and how they are connected and in some cases extended. The forthcoming chapter will present a case that is implemented using the framework.

4. Case Study

The purpose of this chapter is to describe an area with the need for mobile information systems and use our framework to design a supporting multi-interface mobile system.

The structure of this chapter is:

1. Create an as-is task model and present it to identify shortcomings of the current situation.
2. Create a to-be task model evolved from the As-Is Task Model.
3. Create an abstract dialogue model that reflects the to-be task model using our set of abstract dialogue patterns.
4. Develop concrete dialogue models which are realized to the relevant platforms.

4.1 The Medication System

Managing medications is an important task of most health institutions like hospitals, nursing homes and in home care nursing. Our impression is that most existing medication systems don't have a mobile front end despite that many situations demand mobility. Health institutions differ in how they are run and what type of patients they treat. As a consequence the contexts where tasks related to the medication process changes.

4.1.1 Task model

The data gathering for this case study has been done by interviews of Dr. Tomas Norheim Alme. He works at Voss Hospital, Norway and is the entrepreneur of Medicom AS, a company creating mobile solutions for health services. He has been studying the medication routines on several Norwegian hospitals by observations and interviews as part of his work for Medicom AS.

We have made the assumption that the health institutions described in the task model has an existing stationary patient medication system.

As we illustrated earlier in this report, our approach to task modeling is to first model the as-is model and from this envision a to-be model.

The As-Is Task Model

The main tasks focusing on managing medications in a health institution are ordination (start a medication), preparing drugs, handing out drugs to the patients and seponation (stop a medication). It is important to see these tasks in the contexts they are a sub-task of: Ordination and seponation are done in the context of a patient consultation. Preparing and handing out drugs are done in the context of a medication round. This case study will focus on the patient consultation task.

A patient consultation can include many sub-tasks, but we will only include ordination and seponation of medications. Our scope also only includes situations where a nurse is responsible for preparing and handing out medications to the patients.

Ordination and seponation is done by a doctor or by a nurse on behalf of a doctor. A doctor or nurse is generally a female or male between 24 and 67 years old and we can assume they have some computer experience since the institution already use a stationary medication system.

Ordination means setting a patient on a drug by creating a medication entity. A medication contains an ordination date (start date), the ordinating doctor, an optional seponation date (end date), the seponating doctor, a drug, form (tablets, liquid etc.), dosage and optional instructions.

Seponation means stopping an ongoing medication by setting or changing a seponation date and seponating doctor on an existing medication. This can be done if the medication does not contain a seponation date or when the doctor wants to change an existing seponation date for some reason.

Both tasks are triggered by the doctor's consultation based on the patient's historical data like records, lab results, existing and former medications, and by examining the patient's current physical and mental condition. The complete process is illustrated in (Figure 18).

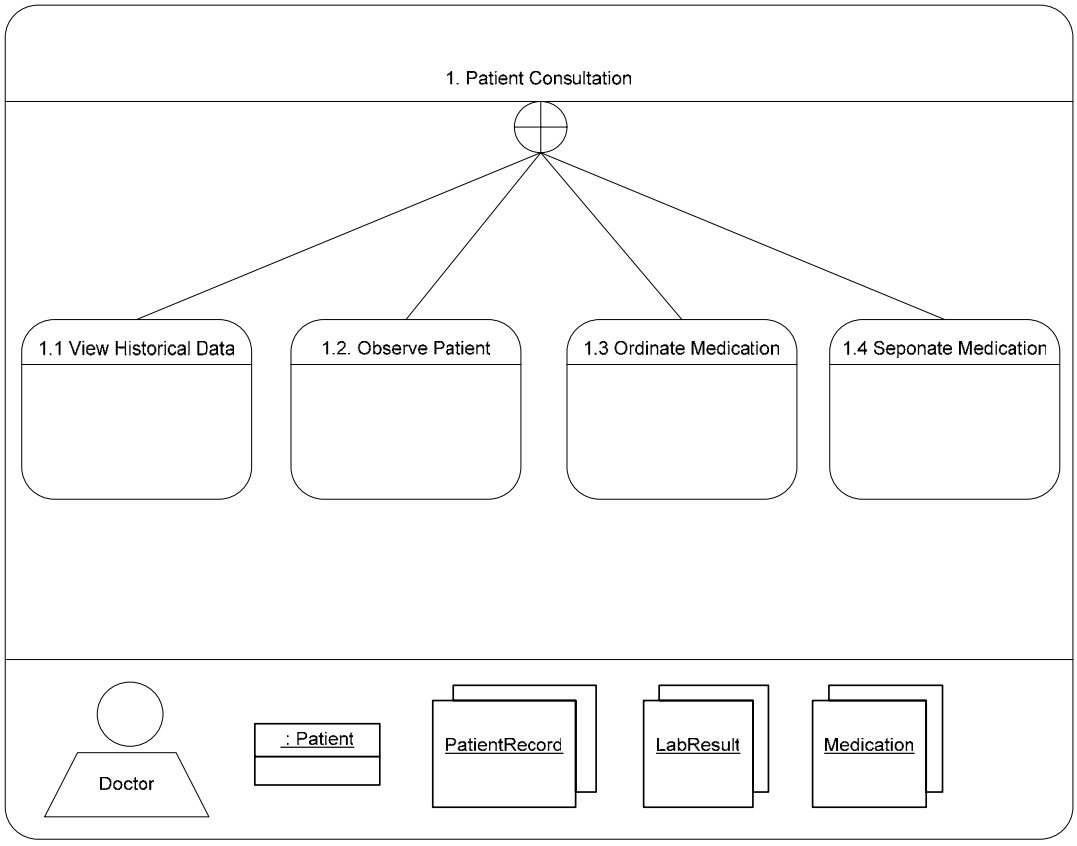

Figure 18: Patient Consultation Task Model

We will from now on, in this case study, isolate the medication tasks to simplify the scope.

This is a quite straight forward process and for an in-office doctor consultation this is the situation because the office is equipped with a pc where the doctor can access all medication data for the patient. The situation is not quite that simple in other situations, for example on a doctor's consultation at the patient's home. In this case the doctor will have to bring with them all the information needed to carry out the consultation and make notes on any ordinations and seponations, and then after the consultation when they is back in their office the doctor or a secretary will have to register the ordinations and seponations into the medication system (Figure 19).

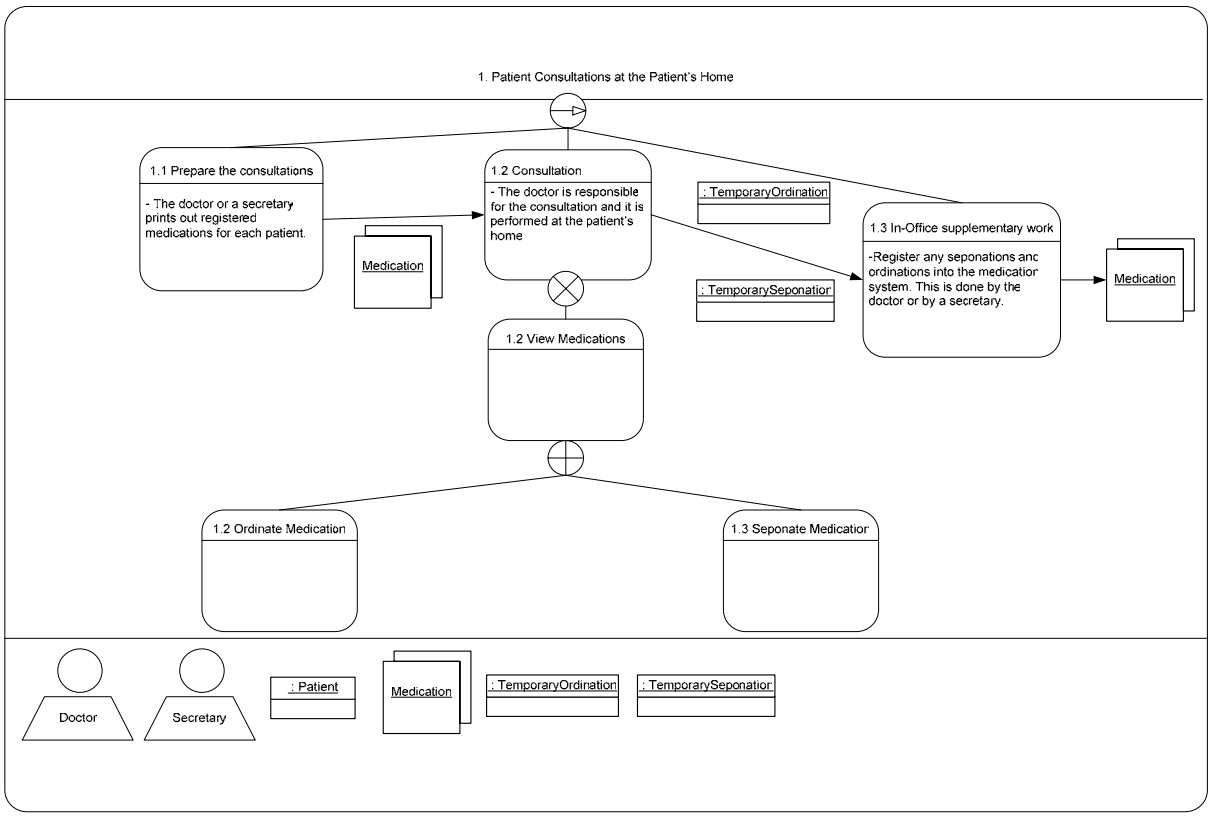

Figure 19: Patient Consultation Task Model 2

The analysis of the user context is separated for each of the three tasks '1.1 Prepare the Consultations', '1.2 Consultations' and '1.3 In-Office Supplementary Work':

- The '1.1 Prepare the Consultations' task is an in-office task so there are no changes to the spatial context. The task is sequential and temporal context only demand for monochronicity. Since it is a traditionally in-office task environmental and social context is not relevant.

- The ‘1.2 Consultations’ task is performed at each patient’s home and therefore has changing spatial context. This demand support for remote mobility, but their route is predictable and planned and their tasks are only business related so they can be categorized as pilgrim users. None of the sub-tasks are performed in parallel so temporal context is monochromic. We can not do any assumptions on environmental characteristics other than that the patient’s home is an apartment or a house.
- The ‘1.3 In-Office Supplementary Work’ task’s user context is equal to the user context of ‘1.1 Prepare the Consultations.

The situation is similar for a doctor’s round in health institutions like a hospital. In this setting the typical sequence is that a nurse makes the preparations for the round by printing out medication records for each patient. The doctor observers the patient and makes the decision for any ordinations or seponations, but asks a nurse to effectuate it. The nurse will then make a note and register the ordinations and seponations in the system after the round (Figure 20).

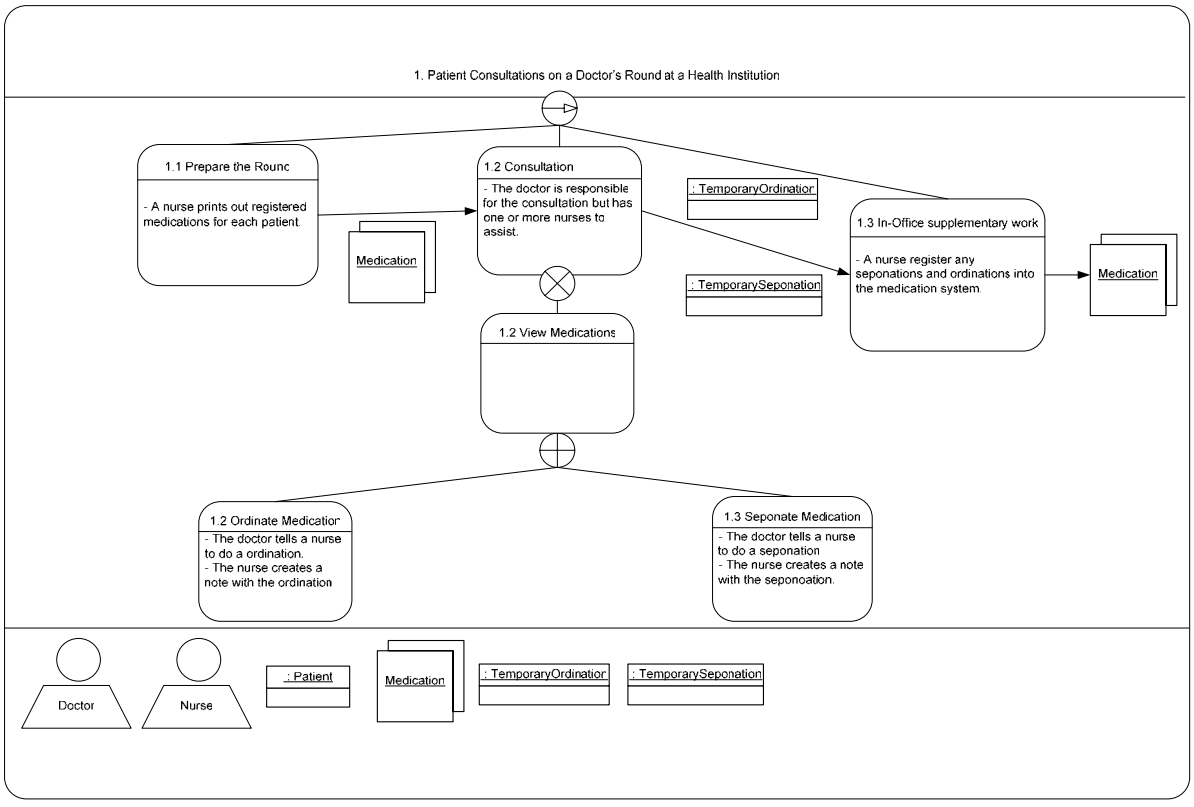

Figure 20: Patient Consultation Task Model 3

The analysis of the user context is separated for each of the three tasks ‘1.1 Prepare the Round’, ‘1.2 Consultations’ and ‘1.3 In-Office Supplementary Work’:

- The ‘1.1 Prepare the Round’ tasks is an in-office task so there are no changes to the spatial context. The task is sequential and temporal context only demand for monochronicity. Since it is a traditionally in-office task environmental and social context is not relevant.
- The ‘1.2 Consultations’ task is performed at each patient’s bed in a hospital and therefore has changing spatial context. This demands support for local mobility. None of the sub-tasks are performed in parallel so temporal context is monochromic. There are no special environmental characteristics to consider.
- The ‘1.3 In-Office Supplementary Work’ tasks’ user context is equal to the user context of ‘1.1 Prepare the Round.

Both of these two scenarios are quite similar because both require a preparation task where all medication records need to be printed out and both are based on making temporary seponations and ordinations for later doing the registration in the medication system. The differences are in the user contexts for the ‘Consultations’ tasks.

From now on in this case study we will focus on the consultations at patient’s home scenarios.

The To-Be Task Model

The goal of the new system is to make the doctor able to view, ordinate and seponate medication directly into the medication system where the patient consultation take place (in this context; the patient’s home). This will completely eliminate the ‘1.1 Prepare the Consultations’ task, the ‘1.3 In-Office Supplementary Work’ task and the nurse actor from the process. The system will be an online distributed system so the nurse team doing the medication round will have any changes earlier and might be able to effectuate them straight away. The new system will therefore be of some organizational support, but is mainly meant to be of personal support.

To meet these goals we will create a mobile front end on the existing medication system. It will run on mobile phones since most people are in possession of such a device and most areas have GSM/GPRS coverage. This will also reduce the infrastructure costs. The characteristics of mobile phones are that it either has a rectangular portrait mode screen with widths from round 128-176 pixels and heights round 128-208 pixels, or a rectangular landscape mode screen with widths round 640 pixels and heights round 200 pixels. For input they have a mobile phone keyboard, but some also have a stylus pen and/or a QWERTY keyboard. We will in this case study only support mobile phones with a portrait mode screen and a mobile phone keyboard.

The main development platform will be Java and J2ME, but since not all mobile phones have Java support we will also create a speech based user interface running on a server that the doctor can call. The goal of evaluating our approach has naturally also affected the platform selection, and these two platforms are very diverse and we think they are good candidates to demonstrate the approach.

The user contexts for the new systems will be the same as for the ‘1.2 Consultations’ task from the As-Is Model, but with some additions: The environmental characteristics will be extended with the assumption of GPRS/GSM support and that the doctor has a mobile phone available. The system should also be developed to support navigation while walking. E.g. the doctor will find the next patients medications on her way from the car and to the patient’s house. This influences the physical personal context because the body’s movements while walking will make it harder to do tasks that require precision, e.g. using a stylus pen on a small graphical user interface.

The following task descriptions and models illustrate the new mobile front end:

Task 1: Patient Consultation (super task) (Figure 21)

Goal: The patient's medications are handled.

Actors: Doctor

Scenario description:

1. Set a active patient (Task 1.1)
2. Select between the tasks View Medication (Task 1.2) or Ordinate (Task 1.3)

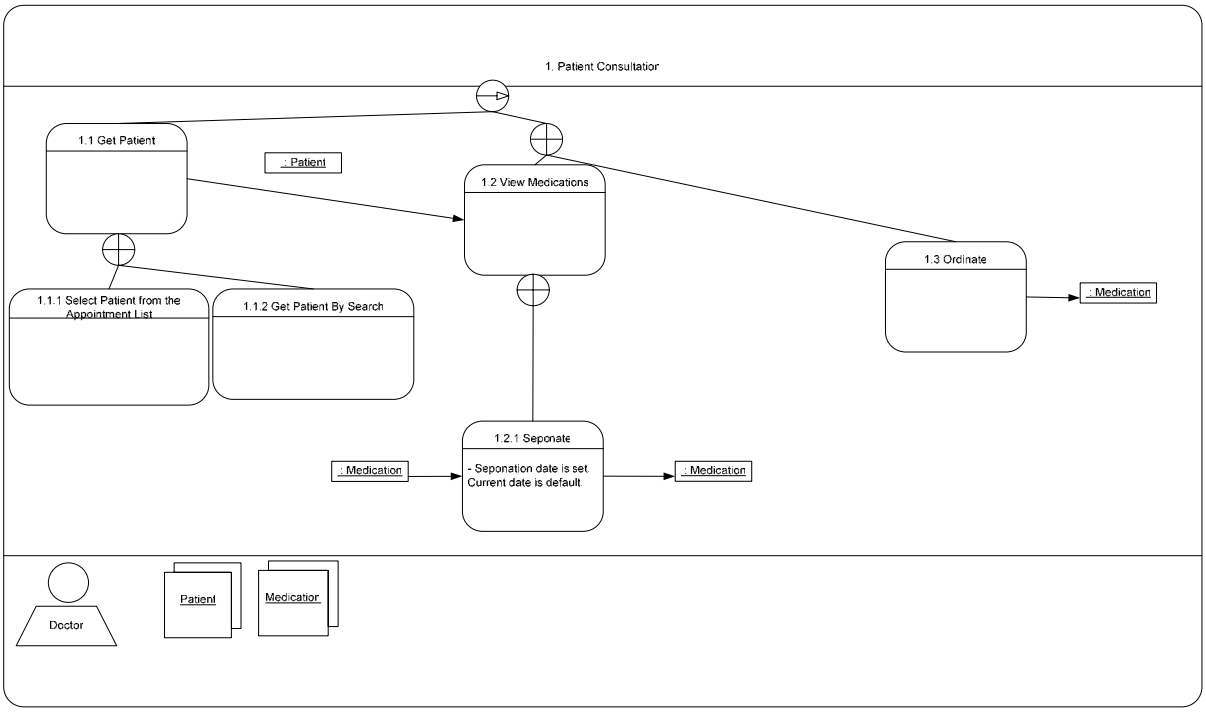

Figure 21: Patient Consultation Task Model 4

Task 1.1: Get Patient

Goal: Set active patient.

Actors: Doctor

Post-conditions: An active patient is set.

Scenario description

1. Task 1.1.1 Select Patient from Appointment List or 1.1.2 Get Patient by Search.

Task 1.1.1: Select Patient from Appointment List (Figure 22)

Goal: Get a patient by selecting an appointment from a appointment list

Actors: Doctor

Scenario description

1. Select a date. Current date is default.
2. Retrieve a list of appointments for the specified date.
3. Select an appointment where the patient is extracted from.

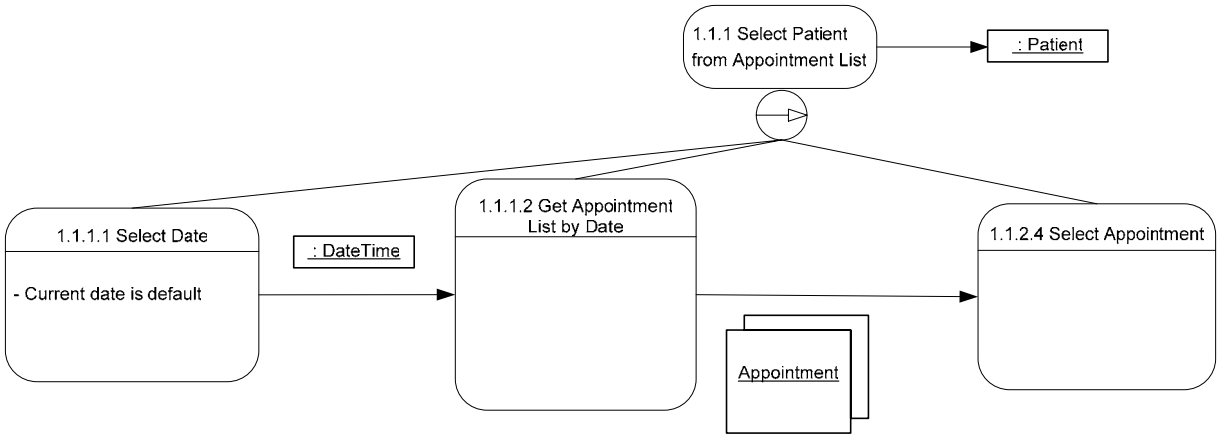

Figure 22: Select Patient from Appointment List

Task 1.1.2: Get Patient by Search (Figure 23)

Goal: Retrieve a list of patients based on name or social security number matching the query.

Actors: Doctor

Scenario description

1. Select search type. ‘Name’ or ‘SSN’.
2. Fill in query
3. Retrieve list of patients
4. Select a patient from the list

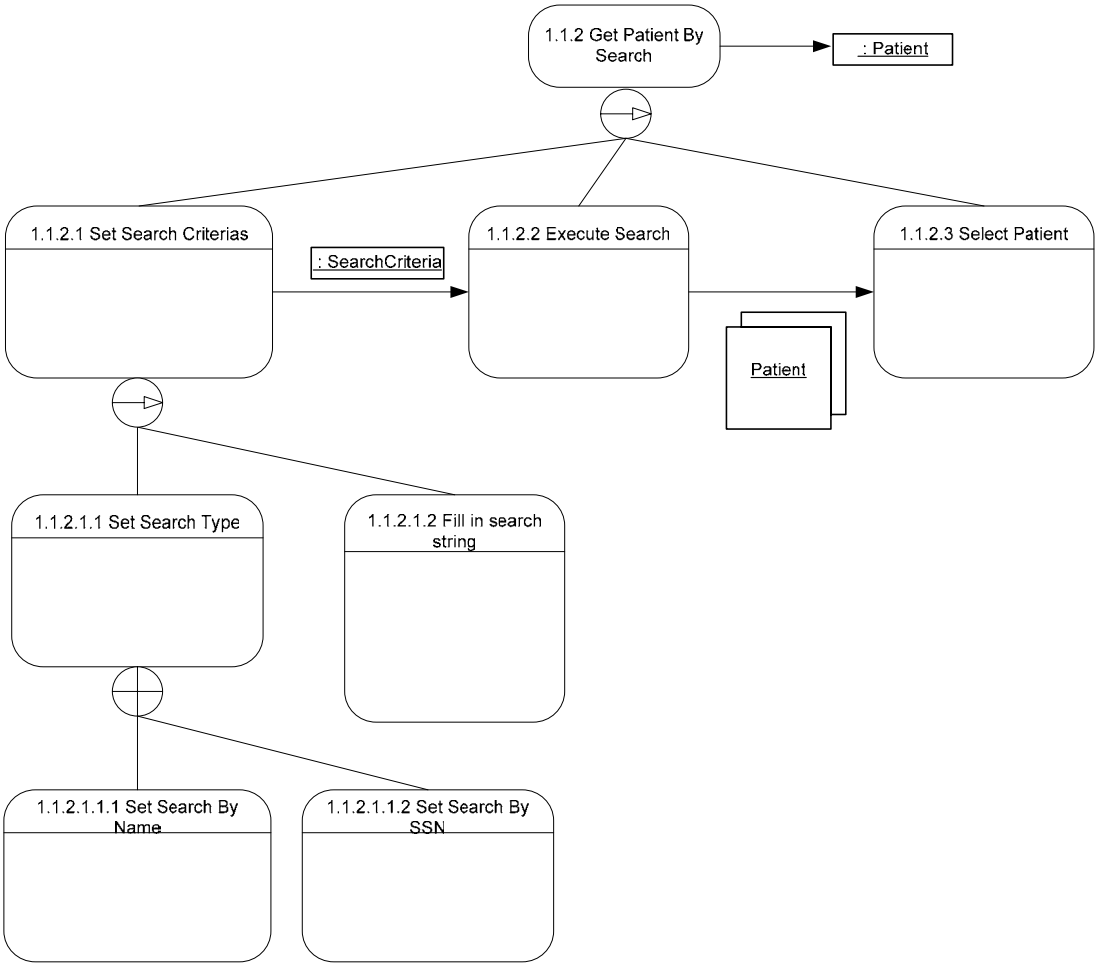

Figure 23: Get Patient by Search

Task 1.2: View Medications

Goal: Retrieve a list of a selected patient's medications. This list should be sorted by date.

Actors: Doctor

Pre-conditions: A patient is selected (Task 1.1).

Scenario description

1. Retrieve a list of all medications.
2. A medication can be selected to perform a seponation. (Task 1.2.1)

Task 1.2.1: Seponate

Goal: Stop a medication from a specified date.

Actors: Doctor

Pre-conditions: A medication is selected (Task 1.2).

Scenario description

1. Set the seponation date on a medication. Current date is default.

Task 1.3: Ordinate (Figure 24)

Goal: Create a new medication on the selected patient.

Actors: Doctor

Pre-conditions: A patient is selected (Task 1.1).

Post-conditions: A new medication is created on the patient.

Scenario description

1. Find a drug. (Task 1.3.1). This is done by traversing the Anatomical Therapeutic Chemical (ATC) [23] hierarchy. The ATC hierarchy classify drugs and divide them into different groups according to the organ or system on which they act and their chemical, pharmacological and therapeutic properties. The hierarchy consists of five levels where the 5th level is the chemical substance.
2. Fill in details about the medication. (Task 1.3.2)

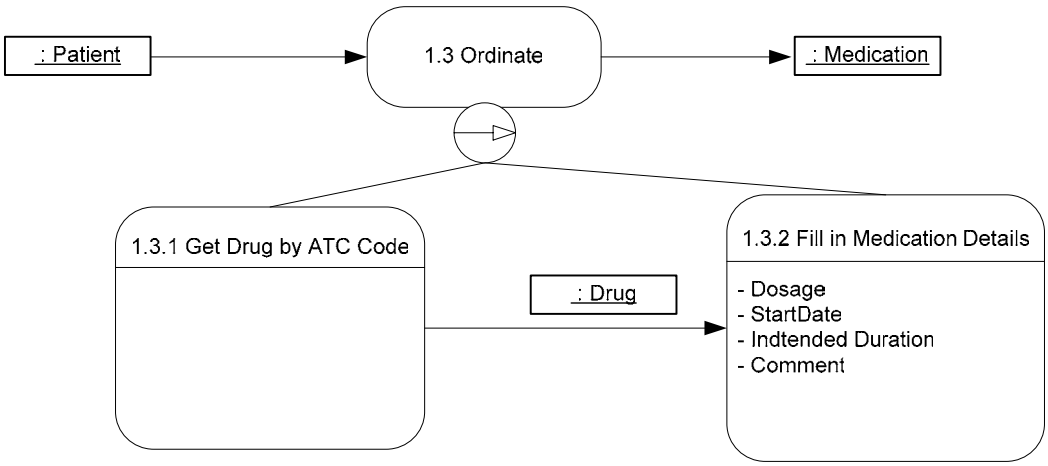

Figure 24: Ordinate

Task 1.3.1: Get Drug by ATC Code (The ATC system is explained in Task 1.3)

Goal: Select a drug from a hierarchy based on ATC codes.

Actors: Doctor

Scenario description

1. Get all top level ATC elements
2. Select a element
3. Get all child elements and return to step 2 if the element is a composite element.
4. Return the drug if it is a leaf element.

Task 1.3.2: Fill in Medication Details

Goal: Create a medication and fill in all essential information.

Actors: Doctor

Pre-conditions: A drug is selected

Scenario description

1. A new medication is created and the selected drug is associated to it.
2. Fill in essential information like dosage, intended duration and comments.

From these models and the to-be task model we have evolved the initial data model in (Figure 25).

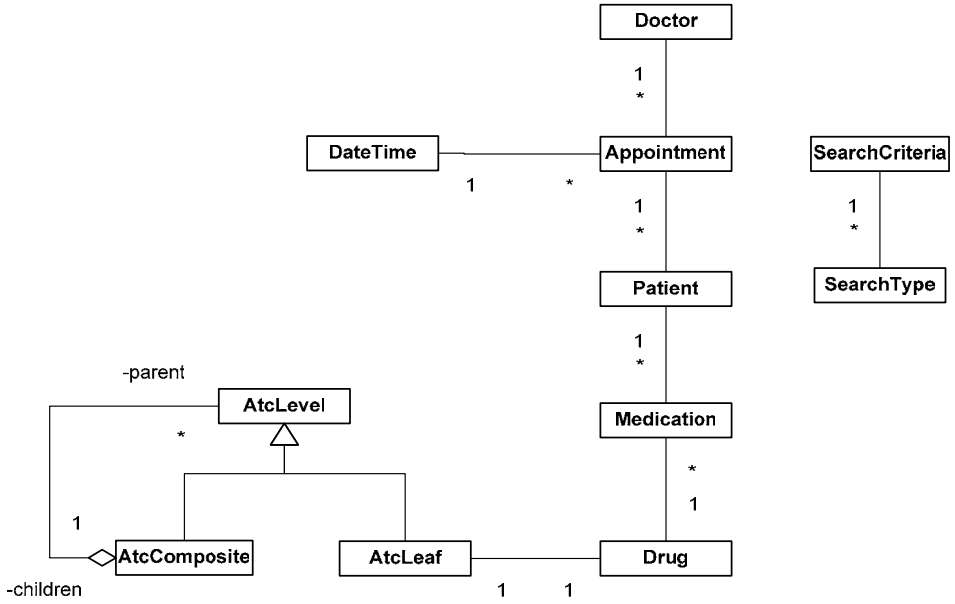

Figure 25: Initial Data Model

4.1.2 Abstract Dialogue Models

The abstract dialogue models are directly mapped from the to-be-task models and they are also created using our set of abstract patterns. These patterns are specified using the associated stereotype.

The first model (Figure 26) illustrates the main flow of the system where the 'Interactor selection' pattern is used to illustrate this. This is specified using the stereotype <<interactorselection>>. We have made the assumption that all interactors give the user the opportunity to go back to the former interactor so this is not modeled. Note that the task '1.2.1 Seponation' from the task model is not reflected in this model. This is because it is triggered by the task '1.2 View Medications' and this is handled by the Edit/View Element pattern which also can handle all the operations needed in the seponation task.

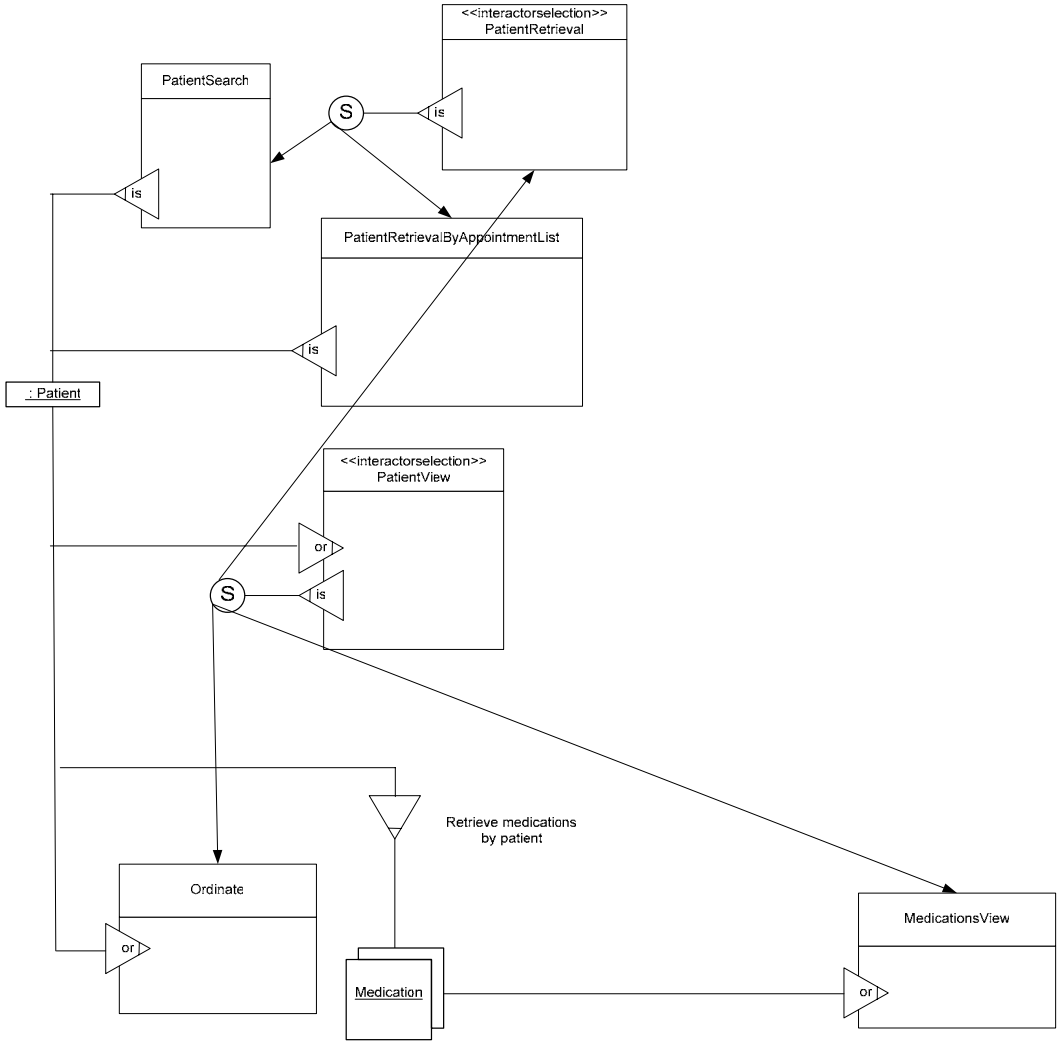

Figure 26: Main flow of the system

The following models are decompositions of the main model. Each interactor which has no stereotype needs to be decomposed.

The PatientRetrievalByAppointmentList interactor is decomposed into a sequence of interactors:

1. An interactor where the user can select a date to filter the appointment list. This can be handled by the element selection pattern. This is specified using the <<elementselection>> stereotype.

2. An interactor where the user can select an appointment from an appointment list filtered by the selected date. This can also be handled by an element selection pattern.

This is illustrated in (Figure 27).

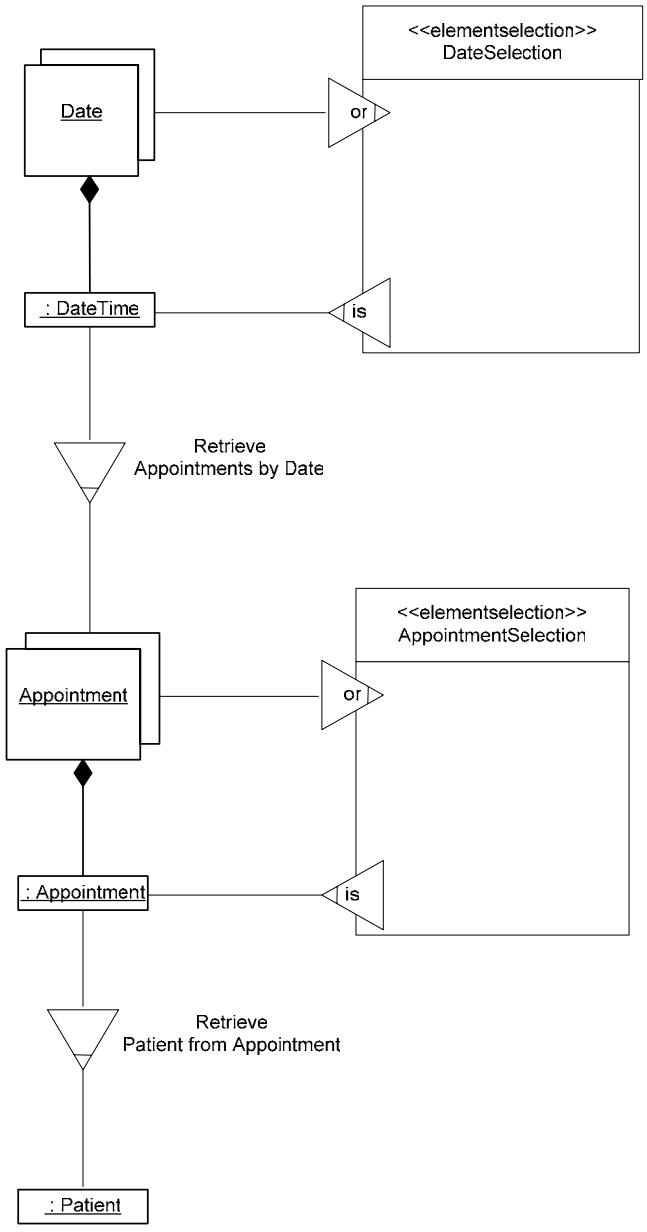

Figure 27: PatientRetrievalByAppointmentList interactor

The PatientSearch interactor is decomposed into a sequence of interactors:

1. An interactor where the user can select a search type. The search type can be either ‘name’ or ‘ssn’ (social security number). This can be handled by the element selection pattern.
2. An interactor where the search criteria entity can fill be edited. This can be handled by the edit/view pattern. This is specified with the <<elementview>> stereotype.

3. An interactor where the user can select a patient from a patient list. This can be handled by the element selection pattern.

This is illustrated in (Figure 28).

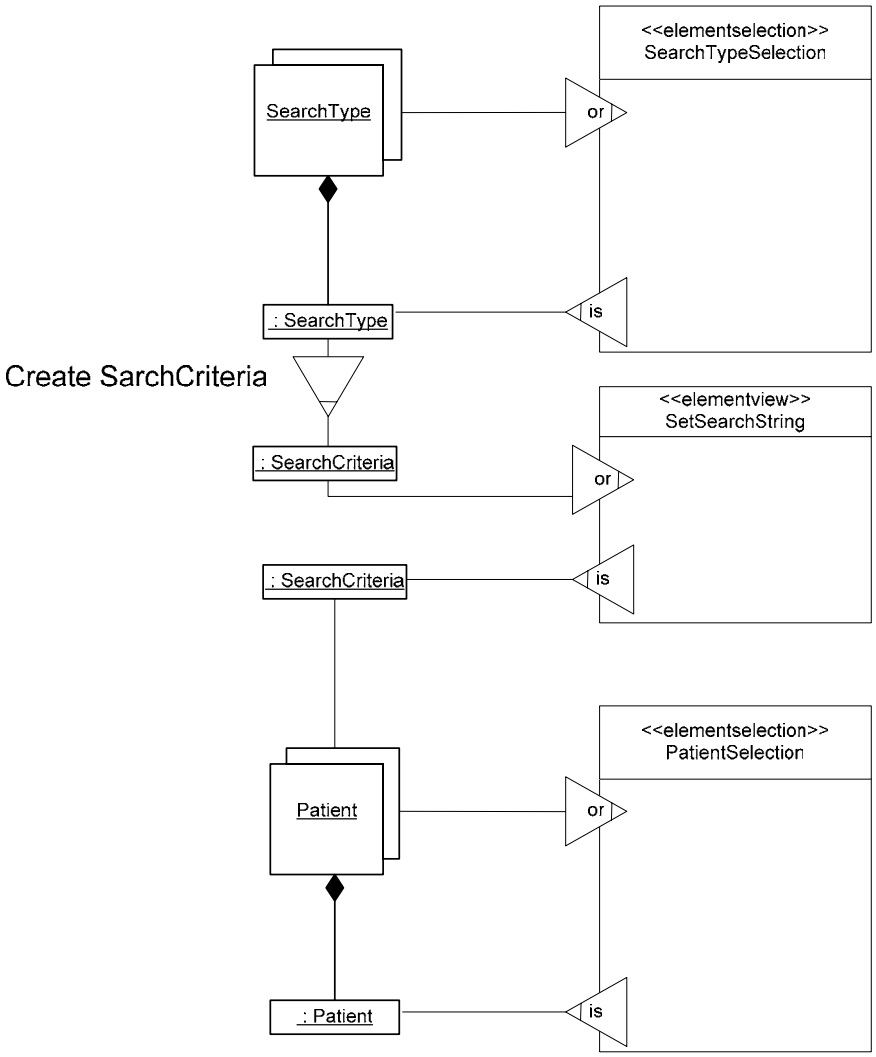

Figure 28: PatientSearch interactor

The Ordinate interactor is decomposed into the following sequence of interactors:

1. An interactor where the user can select a drug from the ATC code hierarchy. The ATC hierarchy is modeled used the composite pattern [13] and can be handled by the hierarchy selection pattern. This is specified using the <<hierarchyselection>> stereotype.
2. An interactor where the medication entity attributes can be edited. This can be handled by the edit/view pattern.

This is illustrated in (Figure 29).

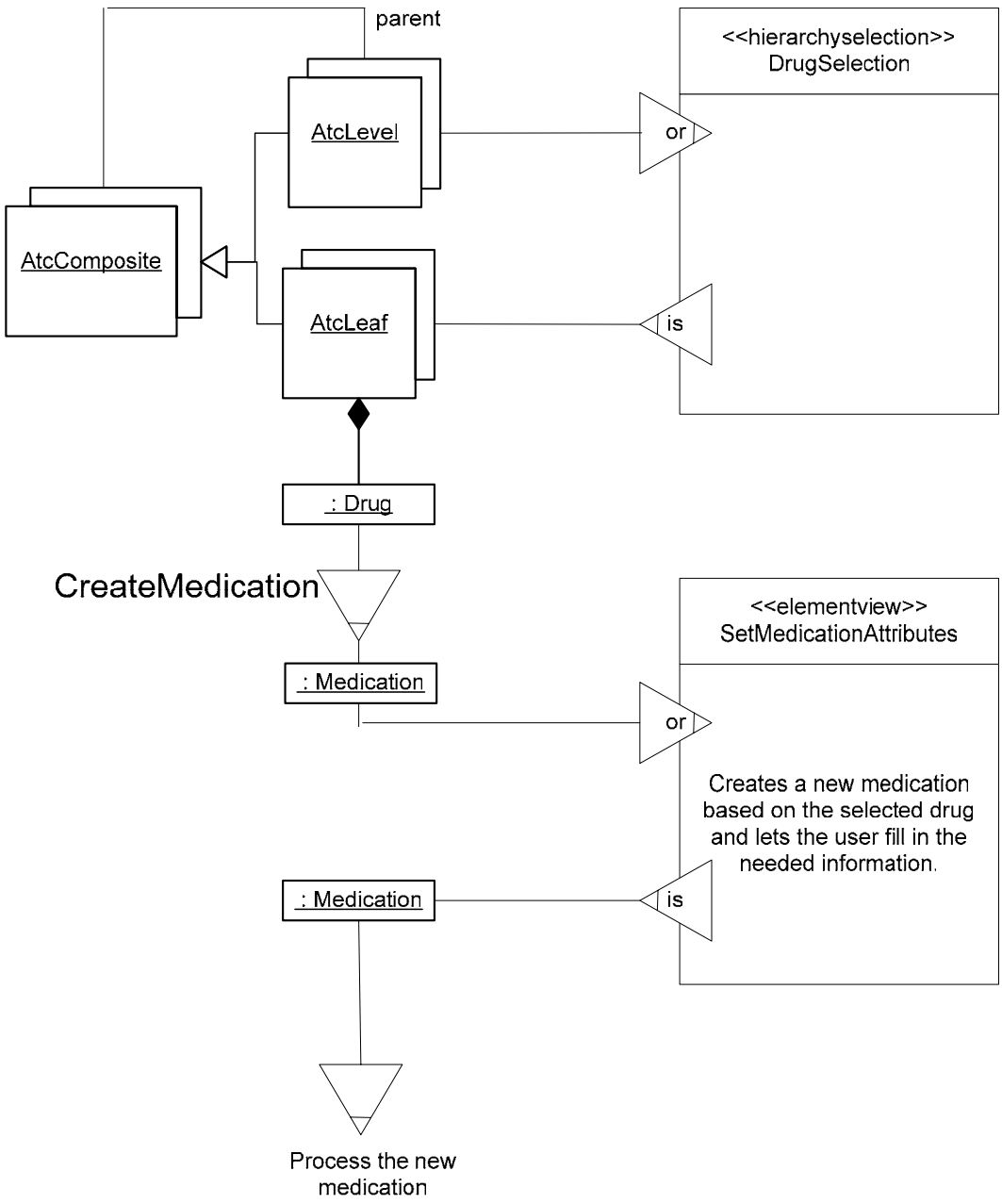

Figure 29: Ordinate interactor

The MedicationsView interactor is decomposed into the following sequence of interactors:

1. An interactor where the user can view a list of all the medications associated to the patient. The user can select a medication from the list. This can be handled by the element selection pattern.
2. An interactor where the user can view the medication entity and if desired set a new seponation date value in a medication entity (the seponating doctor is set by the system). This can be handled by the edit/view element pattern.

This is illustrated in (Figure 30).

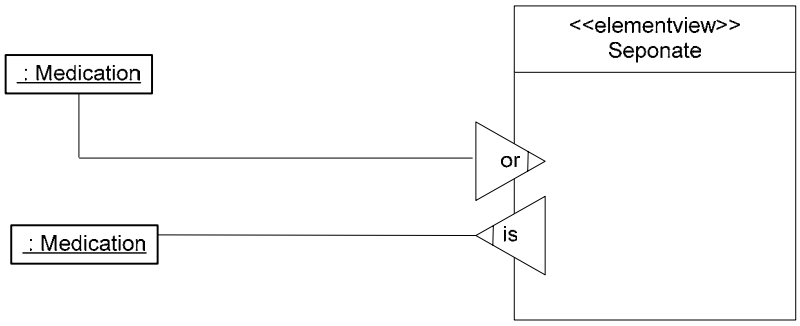

Figure 30: MedicationsView interactor

From this model we see that the following patterns are used: Interactor Selection, Element Selection, Hierarchical Selection and Element View. Each of these patterns has an inherent abstract implemented class including platform independent features and compulsory interface specifications for the platform specific sub classes.

Data Model

The data model is also a part of the abstract modeling. While modeling the abstract dialogue model we have evolved the following data model (Figure 31):

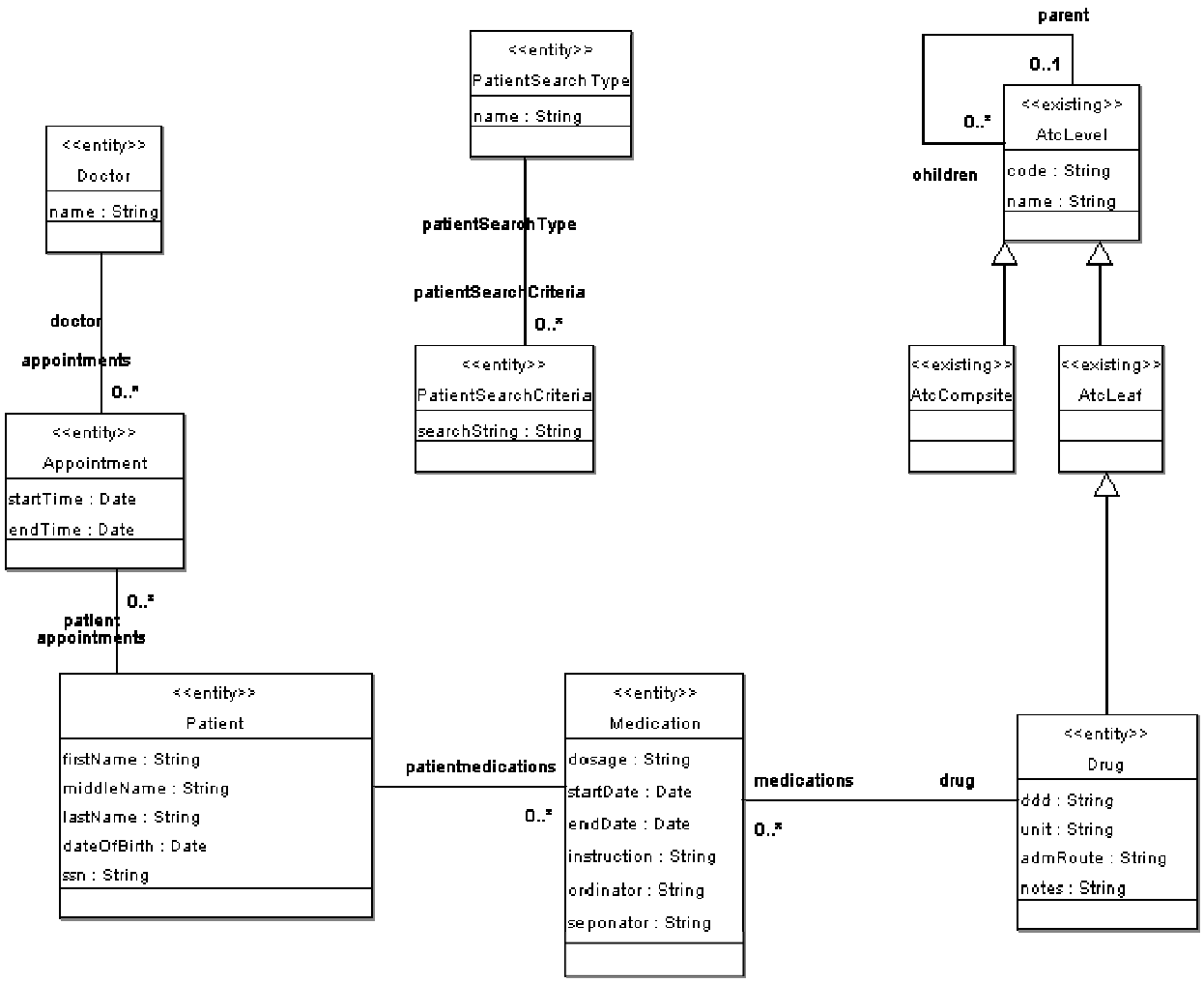

Figure 31: Data model

The data model is fully generated by using the code generator, UmlAPI (see Appendix C). The rule template used to generate to data model code can be found in Appendix D, page 90.

4.1.3 Concrete Dialogue Models

In this case study we will support Java enabled mobile phones and a speech based user interface that for example can run on a server accessible from any phone. All of these concrete interfaces will use the same data model, but have individual specifications for the interactors in the abstract dialogue model. We model these interactor controls as UML classes with the same stereotype as in the abstract dialogue model. Each of these classes will have platform specific specialties expressed in OCL.

Mobile phone specific model

For the mobile phone specific model we have created a set of abstract classes for each pattern that inherit the platform independent abstract class. These classes include all common platform specific features for the specific pattern. When a pattern is used a static class is modeled with the stereotype of the abstract pattern, but it may be specified with platform dependent OCL expressions. These classes will then be run thru the code generator which will generate classes with the OCL expressions transformed into java expressions.

The forthcoming sections will describe how we realized the patterns to the mobile phone platform. The mobile phones must support MIDP 2.0.

Element Selection Realization

In MIDP 2.0 there exist two different user interface components for selecting elements from a collection: List and Choice group. Choice groups are more often known as radio button groups and are best suited for small static collections while Lists can easier handle large dynamic collections. Our mapping rules for this pattern are very simple; all collections with three or fewer elements will be implemented using Choice Groups while a collections length greater than three will use Lists. This is handled in the abstract MIDP specific class. Despite which MIDP user interface control that is used a string describing each individual element will be needed. The abstract MIDP class will therefore specify an abstract method 'public String getDisplayName(Object element)' which will have to be specified in sub classes. This is done as an OCL expression in the platform specific model. An OCL expression defining a method 'public void initialize()' is also needed. This method is called by the constructer in the generated class and is used to initialize attributes in the class. In this pattern the description attribute can be initialized. The description attribute will be shown to the user as an instruction string. The code for the abstract MIDP specific class is presented in Appendix D, page 110.

When this pattern is used and a specific model is specified; the 'UmlAPI' code generator will generate code using an inherent rule template developed for this pattern and platform. The transformation rules for this pattern can be found in Appendix D, page 110. The generated class will be a sub class of the abstract MIDP specific class with the implantation of the getDisplayName method and the initialize method.

Hierarchical Selection

There are no existing interface components that handle hierarchical data structures in MIDP 2.0. In a Java Swing environment this could be handled by a tree view control. We have therefore chosen to solve this by creating a composite control that uses the element selection control. Each level of the hierarchy will be presented to the user using the element selection control. The user can select an element; if it is a composite element the element's child elements will be presented, and if it is as leaf element the control will perform callbacks. Since we are using the element selection control the getDisplayName method must be specified as an OCL expression. An OCL expression defining a method 'public void initialize()' is also needed. This method is called by the constructer in the generated class and is used to initialize attributes in the class. In this pattern the description attribute can be initialized. The code for the abstract MIDP specific class is presented in Appendix D, page 115, but is just an empty shell because there is not any generic code for this control.

When this pattern is used and a specific model is specified the 'UmlAPI' code generator will generate code using an inherent rule template developed for this pattern and platform. The transformation rules for this pattern can be found in Appendix D, page 114.

Edit/View Entity

The Edit/View Entity pattern is a pattern which normally does not has a correspondent control in a user interface development platform and this is also the case for MIDP 2.0. We solve this by creating a composite control where each attribute is handled by a child control. This is based on the approach proposed by [24] where a graphical user interface is generated directly from the data model. This approach lets a single data model be mapped into multiple user interfaces by substituting the rule set. Such an approach suits our problem definition well, but it has a very important shortcoming: It does not provide support for modeling the interaction structure. This is especially important for providing the relation to the task model, but it has a good set of mapping rules from data objects to GUI interaction objects and this is what the Edit/View Entity pattern is all about.

Each attribute of a primitive type is handled by an associated set of mapping rules and controls. We use the primitive data types defined in OCL 2.0 [19] which are Boolean, Integer, Real and String, in our data models. Each of these types will have related user interface components for each supported platform. In many cases the attribute type definition does not contain enough information to find the best conforming user interface component. Attributes definitions should therefore be extended with inherent information to support this. This can be invariants like number of digits (length) in a Real, maximum number of characters in a String or value range on an Integer. Other metadata to clarify the intended semantics can also be included like precision. This information is not supported in the current version of the framework, but each domain object attribute has inherent attributes in the control class that indicates if the domain attribute is visible (Boolean <attribute name>Visible), read-only (Boolean <attribute name>Enable), which order it should be presented (Integer <attribute name>Index), and a description string (String <attribute name>Label). An OCL expression defining a method ‘public void initialize()’ must be specified and is used to initialize these attributes. This method is called by the constructer in the generated class. We assume that this information can be different for each platform, for example a description string may need to be abbreviated for some platforms, and is therefore only specified in the concrete model. This information and the attribute type are used with the following mapping rules:

Boolean

Mapping rules for Boolean are always mapped to check boxes. A check box can have the states read-only or read-writeable. This is handled by the jinteractinocontrol.midp.MIDPBooleanToInteractor class (Appendix D, page 101) and the jinteractorcontrol.midp.MIDPToBooleanInteractor class (Appendix D, page 101).

Integer or Real

Selection of user interface components for numeric types are always mapped to a text field. A text field can have the states read-only or read/writable. The implementation for integer is handled by the jinteractorcontrol.midp.MIDPIntegerToInteractor class (Appendix D, page 105) and jinteractorcontrol.midp.MDIPToIntegerInteractor class (Appendix D, page 106) .

String

Mapping rules for String are always mapped to a text field. A text field can have the states read-only or read/writable. This is handled by the jinteractinocontrol.midp.StringInteractor class. This is handled by the jinteractinocontrol.midp.MIDPStringToInteractor class (Appendix D, page 108) and the jinteractorcontrol.midp.MIDPToStringInteractor class (Appendix D, page 109).

The Edit/View Element control has no generic information for this platform so there is no MIDP specific class. When this pattern is used and a specific model is specified the 'UmlAPI' code generator will generate code using an inherent rule template developed for this pattern and platform. The transformation rules can for this pattern can be found in Appendix D, page 119.

Interactor Selection

In MIDP 2.0 menu structures are mainly made by adding command objects to frame objects, and this is the way we do it in our MIDP specific interactor selection control. The abstract MIDP specific class has a reference to the active form and adds a MIDP command object for each jinteractorcontrols.Command added. Each command needs to be modeled as a class with the <<command>> stereotype. The initialize method must be defined thru an OCL expression where the name attribute is set. These modeled commands will generate a class that inherits jinteractorcontrols.Command. The code for the abstract MIDP specific interaction control class is presented in Appendix D, page 117.

When this pattern is used and a specific model is specified the 'UmlAPI' code generator will generate code using an inherent rule template developed for this pattern and platform. The rule templates for the <<interactorselection>> stereotype and for the <<command>> stereotype can be found in Appendix D, page 116.

Case study model elements

Each of the model elements from the abstract model is modeled in the concrete model as a static class with the inherent stereotype and OCL expressions specifying the platform specific characteristics. The forthcoming sections will describe the platform specific specifications and the generated user interface.

PatientRetrieval

This class does not need any platform specific expressions but it is dependent on a set of Command classes. (Figure 32) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 169.

Stereotype: <<interactorselection>>

Associated command classes:

- PatientSearchCommand
- PatientRetrievalByAppointmentCommand

Figure 32: Patient Retrieval Interactor Phone Screen Shot

PatientSearchCommand

This class will need the *initialize* method to be expressed. This method will initialize the *name* attribute which in this case is: “Patient Search”. The #space# expression is used to indicate a space. For this platform the string length on the *name* attribute should be maximum 20 characters to fit into the screen. The generated source code can be viewed in Appendix D, page 169.

Stereotype: <<command>>

OCL expressions:

- *context medication::ui::midp::PatientSearchCommand def: initialize():Void = Tuple{name = "Patient #space# Search"}*

PatientRetrievalByAppointmentCommand

This class will need the *initialize* method to be expressed. This method will initialize the *name* attribute which in this case is: “Appointments”. The generated source code can be viewed in Appendix D, page 170.

Stereotype: <<command>>

OCL expressions:

- *context medication::ui::midp::PatientRetrievalByAppointmentCommand def: initialize():Void = Tuple{name = "Appointments"}*

DateSelection

This class will need the *initialize* method and the *getDisplayName* to be defined. The *initialize* method initializes the description attribute which in this case is set to: “Select a Date:” This attribute is used to mediate an instruction of the interactor to the user and the value’s string length should be maximum 20 characters long. The *getDisplayName* method will express how to retrieve and manipulate the attributes of the entity to present a descriptive label to the user. In this case the *toDateString* method is used on the *jinterctorcontrols::Date* class. This class is created to simplify and standardize representations of dates in Java since J2ME lack some of the possibilities that exist in J2SE. (Figure 33) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 162.

Stereotype: <<elementselection>>

OCL expressions:

- *context medication::ui::midp::DateSelection def: initialize():Void = Tuple{description = "Select #space# a #space# Date:"}*
- *context medication::ui::midp::DateSelection def: getDisplayName(element: oclAny):String = element.oclAsType(jinteractorcontrols::Date).toDateString()*

Figure 33: Date Selection Interactor Phone Screen Shot

AppointmentSelection

This class will need the *initialize* method and the *getDisplayName* to be defined. The *initialize* method initializes the description attribute which in this case is set to: “Select Appointment:” The *getDisplayName* method use the *getStartTime* method and the *getPatient* method from the *Appointment* object. The *getPatient* method returns a *Patient* object and it therefore expresses how to represent the patient object. This is done by the patient’s last name and first name. (Figure 34) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 160.

Stereotype: <<elementselection>>

OCL expressions:

- *context medication::ui::midp::AppointmentSelection def: initialize():Void = Tuple{description = "Select #space# Appointment:"}*
- *context medication::ui::midp::AppointmentSelection def: getDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::Appointment).getStartTime().toTimeString().concat('#space#').concat(element.oclAsType(medication::businessEntities::Appointment).getPatient().getLastName()).concat('#space#').concat(element.oclAsType(medication::businessEntities::Appointment).getPatient().getFirstName())*

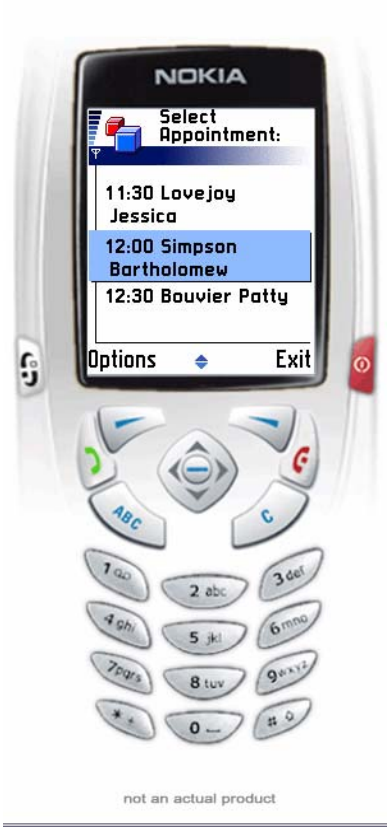

Figure 34: Appointment Selection Interactor Phone Screen Shot

PatientSearchTypeSelection

This class will need the *initialize* method and the *getDisplayName* method to be defined. The *initialize* method initializes the *description* attribute, which in this case is set to: “Select a Search Type:” The *getDisplayName* method use the *getName* method in the *PatientSearchType* object. (Figure 35) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 172.

Stereotype: <<elementselection>>

OCL expressions:

- *context medication::ui::midp::PatientSearchTypeSelection def: initialize():Void = Tuple{description = "Select #space# a #space# Search #space# Type:"}*
- *context medication::ui::midp::PatientSearchTypeSelection def: getDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::PatientSearchType).getName()*

Figure 35: Patient Search Type Selection Interactor Phone Screen Shot

PatientSearchCriteriaView

This class will need the *initialize* method to be defined where the attributes *description*, *patientSearchTypeVisible*, *searchStringEnabled*, *searchStringVisible*, *searchStringIndex, searchStringLabel* is initialized. The string length on the *description* attribute and the label attributes should be maximum 20 characters. All these attributes are generated and is based on the referenced entity class. A method *getPatientSearchTypeDisplay* must also be defined because the *patientSearchType attribute* in the referenced *PatientSearchCriteraView* entity is not a primitive type. (Figure 36) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 170.

Stereotype: <<elementview>>

OCL expressions:

- *context medication::ui::midp::PatientSearchCriteriaView def: initialize():Void = Tuple{description = "Set#space#Query#space# String:", patientSearchTypeVisible = false, searchStringEnabled = true, searchStringVisible = true, searchStringIndex = 0, searchStringLabel = "Query:"}*

- *context medication::ui::midp::PatientSearchCriteriaView def: getPatientSearchTypeDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::PatientSearchType).getName()*

Figure 36: Patient Search Criteria View Interactor Phone Screen Shot

PatientSelection

This class will need the *initialize* method and the *getDisplayName* method to be defined. The *initialize* method initializes the *description* attribute which in this case is set to: “Select a Patient:” The *getDisplayName* method use the *getLastName* method, the *getFirstName* method separated with the ‘,’ character and the *getDateOfBirth* method in the *Patient* object. (Figure 37) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 172.

Stereotype: <<elementselection>>

OCL expressions:

- *context medication::ui::midp::PatientSelection def: initialize():Void = Tuple{description = "Select#space#Patient:"}*

- *context medication::ui::midp::PatientSelection def: getDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::Patient).getLastName().concat(',#space#').concat(element.oclAsType(medication::businessEntities::Patient).getFirstName()).concat('#space#').concat(element.oclAsType(medication::businessEntities::Patient).getMiddleName()).concat('#space#').concat(element.oclAsType(medication::businessEntities::Patient).getDateOfBirth().toDateString())*

Figure 37: Patient Selection Interactor Phone Screen Shot

PatientView

This class does not need any platform specific expressions but it is dependent on a set of Command classes. (Figure 38) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 168.

Stereotype: <<interactorselection>>

Associated command classes:

- OrdinateCommand
- MedicationsSelectionViewCommand

Figure 38: Patient View Interactor Phone Screen Shot

OrdinateCommand

This class will need the *initialize* method to be expressed. This method will initialize the *name* attribute which in this case is: “Ordinate”. The generated source code can be viewed in Appendix D, page 170.

Stereotype: <<command>>

OCL expressions:

- *context medication::ui::midp::OrdinateCommand def: initialize():Void = Tuple{name = "Ordinate"}*

MedicationsSelectionViewCommand

This class will need the *initialize* method to be expressed. This method will initialize the *name* attribute which in this case is: “View Medications”. The generated source code can be viewed in Appendix D, page 169.

Stereotype: <<command>>

OCL expressions:

- *context medication::ui::midp::MedicationsSelectionViewCommand def: initialize():Void = Tuple{name = "View"}*

MedicationSelection

This class will need the *initialize* method and the *getDisplayName* method to be defined. The *initialize* method initializes the *description* attribute which in this case is set to: “Medications:” The *getDisplayName* method use the *getStartDate* method and the *getDrug* method in the *Medication* object. The *getDrug* method returns a *Drug* object and it therefore expresses how to represent the patient object. This is done by the name. (Figure 39) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 168.

Stereotype: <<elementselection>>

OCL expressions:

- *context medication::ui::midp::MedicationSelection def: initialize():Void = Tuple{description = "Medications"}*
- *context medication::ui::midp::MedicationSelection def: getDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::Medication).getStartDate().toDateString().concat('#space#').concat(element.oclAsType(medication::businessEntities::Medication).getDrug().getName())*

Figure 39: Medication Selection Interactor Phone Screen Shot

MedicationVeiw

This class will need the *initialize* method to be defined where the *description* attribute and all the *visible, enable* and *index* attributes for each attribute in the referenced entity class can be initialized. In this case the referenced class is the *Medication* class. Make special notice of that the *endDateEnabled* attribute is not initialized. This is set to *true* as default and that indicates that it is writable and in this case supports the seponation task. (Figure 40) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 164.

Stereotype: <<elementview>>

OCL expressions:

- context medication::ui::midp::MedicationView def: initialize():Void = Tuple{description = "Medication", drugIndex = 0, startDateIndex = 1, endDateIndex = 2, dosageIndex = 3, instructionIndex = 4, ordinatorVisible = false, seponatorVisible = false, dosageEnabled = false, startDateEnabled = false, instructionEnabled = false, drugLabel = "Drug:", dosageLabel = "Dosage:", startDateLabel = "Ordination Date:", endDateLabel = "Seponation Date:", instructionLabel = "Instruction:"}

- context medication::ui::midp::MedicationView def: getDrugDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::Drug).getCode().concat('#space#').concat(element.oclAsType(medication::businessEntities::Drug).getName())

Figure 40: Medication View Interactor Phone Screen Shot

AtcLevelSelection

This class will need the *initialize* method and the *getDisplayName* method to be defined. The *initialize* method initializes the *description* attribute which in this case is set to: “Select a Drug:” The *getDisplayName* method use the *getCode* method and the *getName* method in the *AtcLevel* object. (Figure 41) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 162.

Stereotype: <<hierarchicalselection>>

OCL expressions:

- *context medication::ui::midp::AtcLevelSelection def: initialize():Void = Tuple{description = "Select#space# ATC:"}*

- *context medication::ui::midp::AtcLevelSelection def: getDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::AtcLevel).getCode().concat('#space#'). concat(element.oclAsType(medication::businessEntities::AtcLevel).getName())*

Figure 41: ATC Level Selection Interactor Phone Screen Shots

OrdinateView

This class will need the *initialize* method to be defined where the *description* attribute and all the *visible, enable* and *index* attributes for each attribute in the referenced entity class can be initialized. The referenced class in this case is the *Medication* class. (Figure 42) is a screen shot of the interactor control. The generated source code can be viewed in Appendix D, page 174.

Stereotype: <<elementview>>

OCL expressions:

- *context medication::ui::midp::OrdinateView def: initialize():Void = Tuple{description = "New #space# Medication", drugIndex = 0, startDateIndex = 1, endDateIndex = 2, dosageIndex = 3, instructionIndex = 4, ordinatorVisible = false, seponatorVisible = false, drugLabel = "Drug:", dosageLabel = "Dosage:", startDateLabel = "Ordination Date:", endDateLabel = "Seponation Date:", instructionLabel = "Instruction:"}*
- *context medication::ui::midp::OrdinateView def: getDrugDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::Drug).getCode().concat('#space#').concat(element.oclAsType(medication::businessEntities::Drug).getName())*

Figure 42: Ordinate Interactor Phone Screen Shot

Speech specific model

We have also created a set of abstract classes for speech based user interfaces. This is tailored for the Cloudgarden Talking Java API [25] which offers both text-to-speech and speech-to-text. A big difference from the mobile phone specific classes is that there are no existing user interface components that maps directly to any of our patterns. These classes include all common platform specific features for the specific pattern. When a pattern is used a static class is modeled with the stereotype of the abstract pattern, but it may be specified with platform dependent OCL expressions.

The forthcoming sections will describe how we realized the patterns for the speech platform:

Element Selection

Speech based element selections are presented to the user by first reading out a description text and then all the elements by the index number and an element description text, using text-to-speech. The user can then select an element by saying the index number and/or the description text (speech-to-text). This is handled by an abstract speech specific class. The abstract speech class has specified an abstract method 'public String getDisplayName(Object element)' which is used to extract each elements description string, and an initialize method which is used to set the description string. The description string is used as to describe the feature and an instruction string. The getDisplayName result string should be short and informative. It is very hard to listen to lists with long element identificators. These methods must be specified as OCL expressions in the platform specific model. The code for the abstract speech specific class is presented in Appendix D, page 144.

When this pattern is used and a specific model is specified the 'UmlAPI' code generator will generate code using an inherent rule template developed for this pattern and platform. The transformation rules can be found in Appendix D, page 144. The generated class will be a sub class of the abstract speech specific class with the implantation of the getDisplayName and the initialize methods.

Hierarchical Selection

This control is solved by creating a composite control that uses the element selection control. Each level of the hierarchy will be presented to the user using the element selection control. The user can select an element; if it is a composite element the element's child elements will be presented, and if it is as leaf element the control will perform callbacks. Since we are using the element selection control the getDisplayName method must be specified as an OCL expression. The code for the abstract speech specific class is presented in Appendix D, page 149.

When this pattern is used and a specific model is specified the 'UmlAPI' code generator will generate code using an inherent rule template developed for this pattern and platform. The transformation rules can be found in Appendix D, page 148.

Edit/View Entity

This control is, like the MIDP approach, based on the approach proposed by [24] where a graphical user interface is generated directly from the data model except that we are generating a speech based user interface.

Each attribute of a primitive type is handled by an inherent set of mapping rules and controls. Each domain object attribute has inherent attributes in the control class that indicates if the domain attribute is visible (Boolean <attribute name>Visible), read-only (Boolean <attribute name>Enable), which order it should be presented (Integer <attribute name>Index), and a description string (String <attribute name>Label). We assume that this information can be different for each platform, for example a description string may need to be abbreviated for some platforms, and is therefore only specified in the concrete model. This information and the attribute type are used with the following mapping rules:

<u>String</u>

Mapping rules for String are solved by using text-to-speech for presenting the value for the user. A text field can have the states read-only or read/writable. If the string is writable the user can dictate an input string using speech-to-text in dictation mode. Dictation mode means that no input rules are set and that the built in dictionary is used. This is handled by the jinteractinocontrol.speech.StringToSpeechInteractor class (Appendix D, page 141) and the jinteractorcontrol.speech.ToStringInteractor class (Appendix D, page 142).

Boolean

Boolean values use the StringToSpeechInteractor control for output where the word checked is used for true and the word unchecked for false. If it is writable the speech-to-text feature is used with a defined speech rule that lets the user either user the words ‘yes’ or ‘checked’ for true, or ‘no’, ‘unchecked’ or ‘not checked’ for false. This is handled by the jinteractinocontrol.speech.BooleanToSpeechInteractor class (Appendix D, page 131) and the jinteractorcontrol.speech.ToBooleanInteractor class (Appendix D, page 132).

Integer or Real

Numeric values use the StringToSpeechInteractor control for output where the value is casted to a String. If it is writable the speech-to-text feature is used with a defined speech rule that lets the user say a number. This is handled by the jinteractinocontrol.speech.IntegerToSpeechInteractor class (Appendix D, page 139) and the jinteractorcontrol.speech.ToIntegerInteractor class (Appendix D, page 140).

The Entity/View Element do also have an abstract platform specific class for this platform. This can be viewed in Appendix D, page 161. When this pattern is used and a specific model is specified the ‘UmlAPI’ code generator will generate code using an inherent rule template developed for this pattern and platform. The rule template can be found in Appendix D, page 154.

Interactor Selection

The interactor selection control is handled in the same way as the element selection control except that it only takes jinteractorcontrols.Command objects as input. Each command needs to be modeled as a class with the <<command>> stereotype. The initialize method must be defined through an OCL expression where the name attribute is set. These modeled commands will generate a class that inherits jinteractorcontrols.Command. The code for the abstract speech specific interaction control class is presented in Appendix D, page 151.

When this pattern is used and a specific model is specified the ‘UmlAPI’ code generator will generate code using an inherent rule template developed for this pattern and platform. The rule templates for the <<interactorselection>> stereotype and for the <<command>> stereotype can be found in Appendix D, page 150.

Case study concrete model elements

Each of the model elements from the abstract model is modeled in the concrete model as a static class with the inherent stereotype and OCL expressions specifying the platform specific characteristics.

PatientRetrieval

This class does not need any platform specific expressions but it is dependent on a set of Command classes. (Figure 43) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 208.

Stereotype: <<interactorselection>>

Associated command classes:

- PatientSearchCommand
- PatientRetrievalByAppointmentCommand

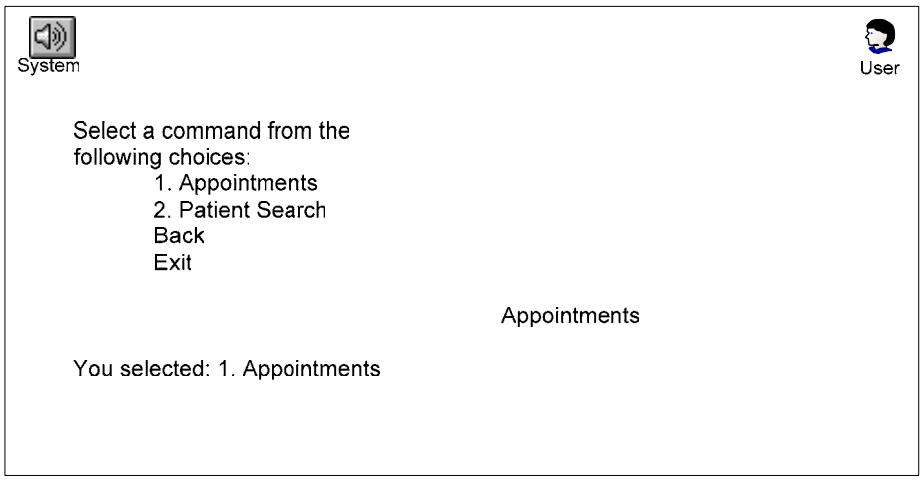

Figure 43: Patient Retrieval interactor

PatientSearchCommand

This class will need the *initialize* method to be expressed. This method will initialize the *name* attribute which in this case is: “Patient Search”. The generated code can be viewed in Appendix D, page 209.

Stereotype: <<command>>

OCL expressions:

- *context medication::ui::phone::PatientSearchCommand def: initialize():Void = Tuple{name = "Patient #space# Search"}*

PatientRetrievalByAppointmentCommand

This class will need the *initialize* method to be expressed. This method will initialize the *name* attribute which in this case is: “Appointments”. The generated code can be viewed in Appendix D, page 209.

Stereotype: <<command>>

OCL expressions:

- *context medication::ui::phone::PatientRetrievalByAppointmentCommand def: initialize():Void = Tuple{name = "Appointments"}*

DateSelection

This class will need the *initialize* method and the *getDisplayName* to be defined. The *initialize* method initializes the description attribute which in this case is set to: “Select a Date:” This attribute is used to mediate an instruction of the interactor to the user. The *getDisplayName* method will express how to retrieve and manipulate the attributes of the entity to present a descriptive label to the user. In this case the *toDateString* method is used on the *jinterctorcontrols::Date* class. (Figure 44) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 182.

Stereotype: <<elementselection>>

OCL expressions:

- *context medication::ui::phone::DateSelection def: initialize():Void = Tuple{description = "Select #space# a #space# Date:"}*
- *context medication::ui::phone::DateSelection def: getDisplayName(element: oclAny):String = element.oclAsType(jinteractorcontrols::Date).toDateString()*

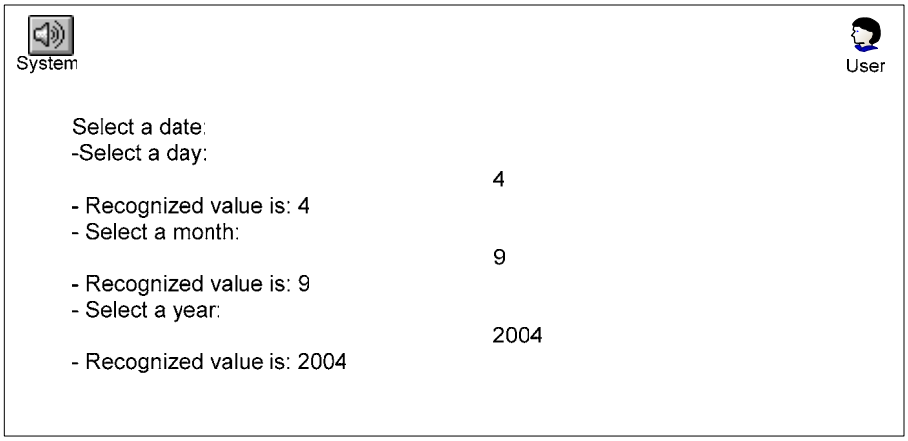

Figure 44: Date Selection interactor

AppointmentSelection

This class will need the *initialize* method and the *getDisplayName* to be defined. The *initialize* method initializes the description attribute which in this case is set to: “Select Appointment:” The *getDisplayName* method use the *getStartTime* method and the *getPatient* method from the *Appointment* object. The *getPatient* method returns a *Patient* object and it therefore expresses how to represent the patient object. This is done by the patient’s last name and first name. (Figure 45) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 180.

Stereotype: <<elementselection>>

OCL expressions:

- *context medication::ui::phone::AppointmentSelection def: initialize():Void = Tuple{description = "Select #space# Appointment:"}*
- *context medication::ui::phone::AppointmentSelection def: getDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::Appointment).getStartTime().toTimeString().concat('#space#').concat(element.oclAsType(medication::businessEntities::Appointment).getPatient().getLastName()).concat('#space#').concat(element.oclAsType(medication::businessEntities::Appointment).getPatient().getFirstName()))*

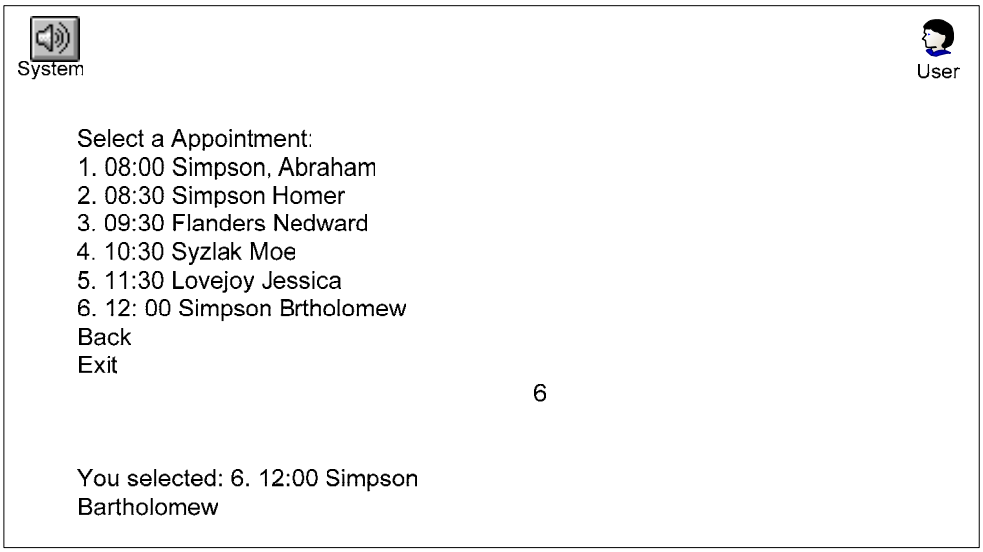

Figure 45: Appointment Selection interactor

PatientSearchTypeSelection

This class will need the *initialize* method and the *getDisplayName* method to be defined. The *initialize* method initializes the *description* attribute which in this case is set to: “Select a Search Type:” The *getDisplayName* method use the *getName* method in the *PatientSearchType* object. (Figure 46) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 213.

Stereotype: <<elementselection>>

OCL expressions:

- *context medication::ui::phone::PatientSearchTypeSelection def: initialize():Void = Tuple{description = "Select #space# a #space# Search #space# Type:"}*
- *context medication::ui::phone::PatientSearchTypeSelection def: getDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::PatientSearchType).getName()*

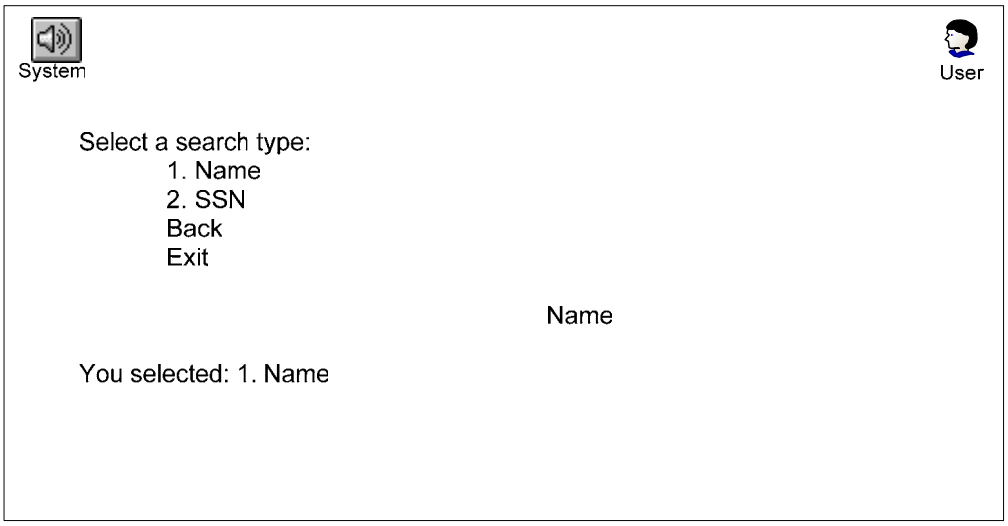

Figure 46: Patient Search Type Selection interactor

PatientSearchCriteriaView

This class will need the *initialize* method to be defined where the attributes *description*, *patientSearchTypeVisible*, *searchStringEnabled*, *searchStringVisible*, *searchStringIndex,* and *searchStringLabel* is initialized. All these attributes are generated and is based on the referenced entity class. A method *getPatientSearchTypeDisplay* must also be defined because the *patientSearchType attribute* in the referenced *PatientSearchCriteraView* entity is not a primitive type. (Figure 47) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 209.

Stereotype: <<elementview>>

OCL expressions:

- *context medication::ui::phone::PatientSearchCriteriaView def: initialize():Void = Tuple{description = "Set #space# Query #space# String:", patientSearchTypeVisible = false, searchStringEnabled = true, searchStringVisible = true, searchStringIndex = 0, searchStringLabel = "Query:"}*
- *context medication::ui::phone::PatientSearchCriteriaView def: getPatientSearchTypeDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::PatientSearchType).getName()*

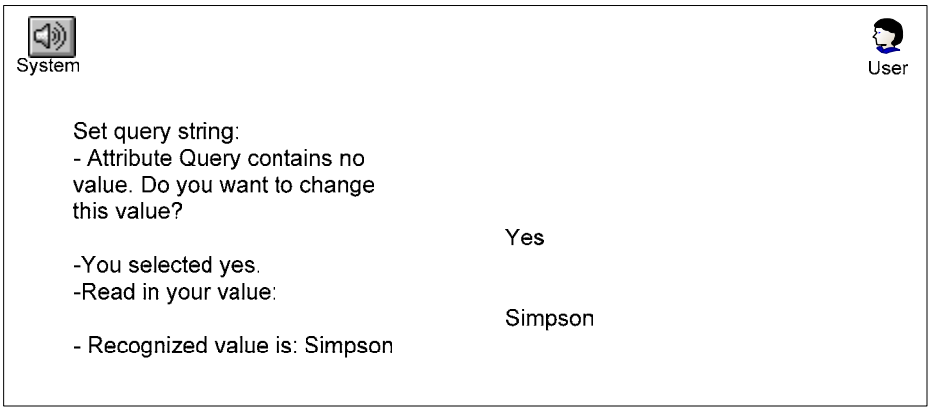

Figure 47: Patient Search Criteria View Selection interactor

PatientSelection

This class will need the *initialize* method and the *getDisplayName* method to be defined. The *initialize* method initializes the *description* attribute which in this case is set to: “Select a Patient:” The *getDisplayName* method use the *getFirstName* method, the *getMiddleName* method and the *getLastName* method in the *Patient* object. Note that this is quite different from the MIDP implementation where the birth date in combination with the name is used as a patient identification. Listening to a list of birth dates is quite hard compared to reading them on a screen so for this platform we only use the name. (Figure 48) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 214.

Stereotype: <<elementselection>>

OCL expressions:

- *context medication::ui::phone::PatientSelection def: initialize():Void = Tuple{description = "Select#space#Patient:"}*
- *context medication::ui::phone::PatientSelection def: getDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::Patient).getFirstName().concat('#space#').concat(element.oclAsType(medication::businessEntities::Patient).getMiddleName()).concat('#space#').concat(element.oclAsType(medication::businessEntities::Patient).getLastName())*

Figure 48: Patient Selection interactor

PatientView

This class does not need any platform specific expressions, but it is dependent on a set of *Command* classes. (Figure 49) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 214.

Stereotype: <<interactorselection>>

Associated command classes:

- OrdinateCommand
- MedicationsSelectionViewCommand

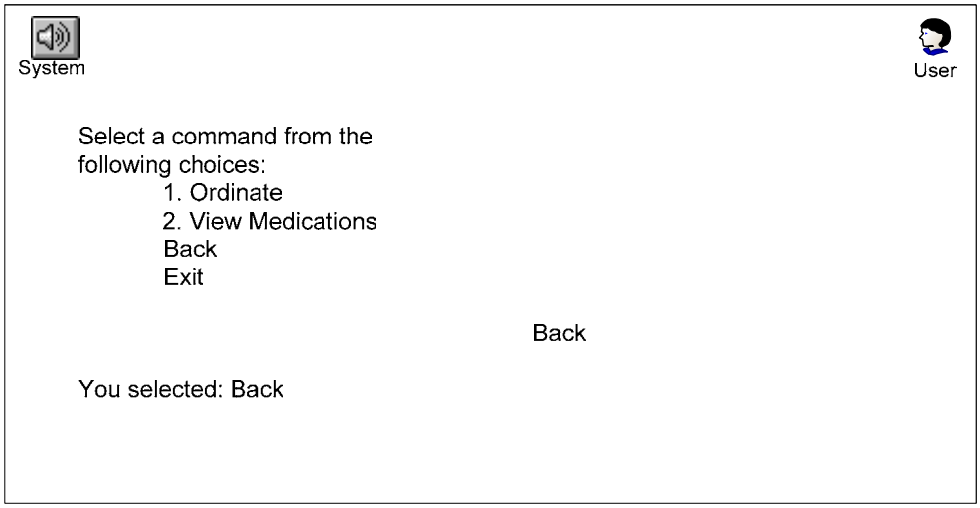

Figure 49: Patient View interactor

OrdinateCommand

This class will need the *initialize* method to be expressed. This method will initialize the *name* attribute which in this case is: “Ordinate”. The generated code can be viewed in Appendix D, page 215.

Stereotype: <<command>>

OCL expressions:

- *context medication::ui::phone::OrdinateCommand def: initialize():Void = Tuple{name = "Ordinate"}*

MedicationsSelectionViewCommand

This class will need the *initialize* method to be expressed. This method will initialize the *name* attribute which in this case is: “View Medications”. The generated code can be viewed in Appendix D, page 215.

Stereotype: <<command>>

OCL expressions:

- *context medication::ui::phone::MedicationsSelectionViewCommand def: initialize():Void = Tuple{name = "View Medications"}*

MedicationSelection

This class will need the *initialize* method and the *getDisplayName* method to be defined. The *initialize* method initializes the *description* attribute which in this case is set to: “The following medications are recorded on the selected patient:” The *getDisplayName* method use the *getStartDate* method and the *getDrug* method in the *Medication* object. The *getDrug* method returns a *Drug* object and it therefore expresses how to represent the patient object. This is done by the name. (Figure 50) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 182.

Stereotype: <<elementselection>>

OCL expressions:

- *context medication::ui::phone::MedicationSelection def: initialize():Void = Tuple{description = "The following medications are recorded on the selected patient:"}*
- *context medication::ui::phone::MedicationSelection def: getDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::Medication).getStartDate().toString().concat('#space#').concat(element.oclAsType(medication::businessEntities::Medication).getDrug().getName())*

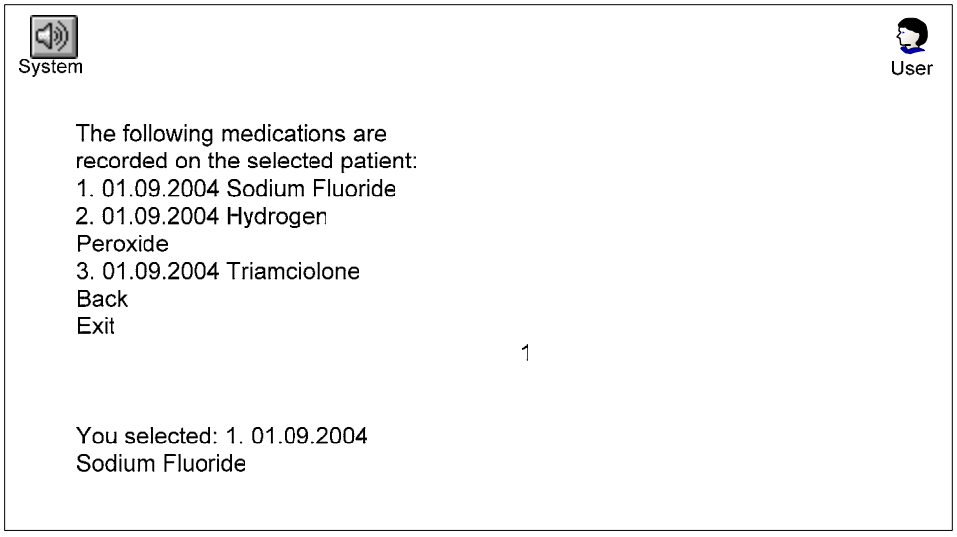

Figure 50: Medication Selection interactor

MedicationVeiw

This class will need the *initialize* method to be defined where the *description* attribute and all the *visible, enable* and *index* attributes for each attribute in the referenced entity class can be initialized. In this case the referenced class is the *Medication* class. Make special notice that the *endDateEnabled* attribute is not initialized. This is set to *true* as default and that indicates that it is writable and in this case supports the seponation task. (Figure 51) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 183.

Stereotype: <<elementview>>

OCL expressions:

- *context medication::ui::phone::MedicationView def: initialize():Void = Tuple{description = "The following data is recorded on the selected medication:", drugIndex = 0, startDateIndex = 1, endDateIndex = 2, dosageIndex = 3, instructionIndex = 4, ordinatorVisible = false, seponatorVisible = false, dosageEnabled = false, startDateEnabled = false, instructionEnabled = false, drugLabel = "Drug:", dosageLabel = "Dosage:", startDateLabel = "Ordination Date:", endDateLabel = "Seponation Date:", instructionLabel = "Instruction:"}*
- *context medication::ui::phone::MedicationView def: getDrugDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::Drug).getCode().concat('#space#').concat(element.oclAsType(medication::businessEntities::Drug).getName())*

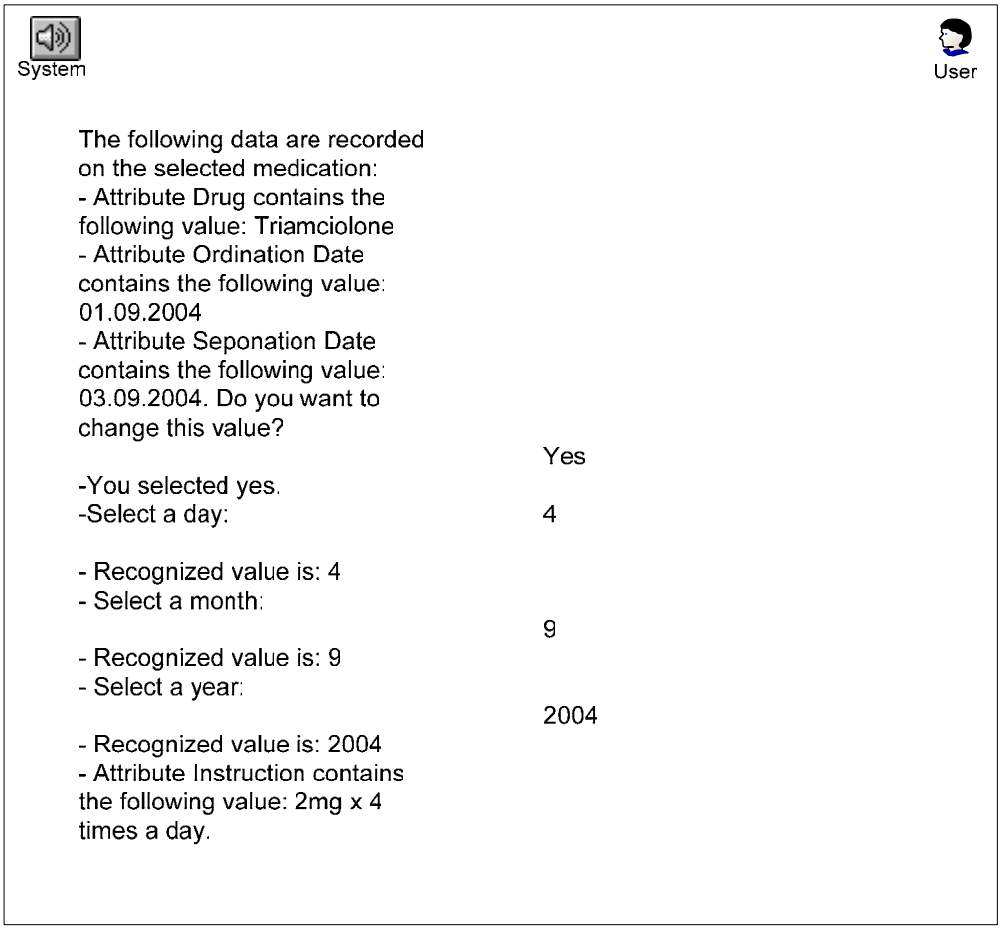

Figure 51: Medication View interactor

AtcLevelSelection

This class will need the *initialize* method and the *getDisplayName* method to be defined. The *initialize* method initializes the *description* attribute which in this case is set to: “Select a Drug:” The *getDisplayName* method use the *getCode* method and the *getName* method in the *AtcLevel* object. (Figure 52) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 180.

Stereotype: <<hierarchicalselection>>

OCL expressions:

- *context medication::ui::phone::AtcLevelSelection def: initialize():Void = Tuple{description = "Select#space#a#space#ATC#space#code :"}*
- *context medication::ui::phone::AtcLevelSelection def: getDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::AtcLevel).getCode().concat('#space#'). concat(element.oclAsType(medication::businessEntities::AtcLevel).getName())*

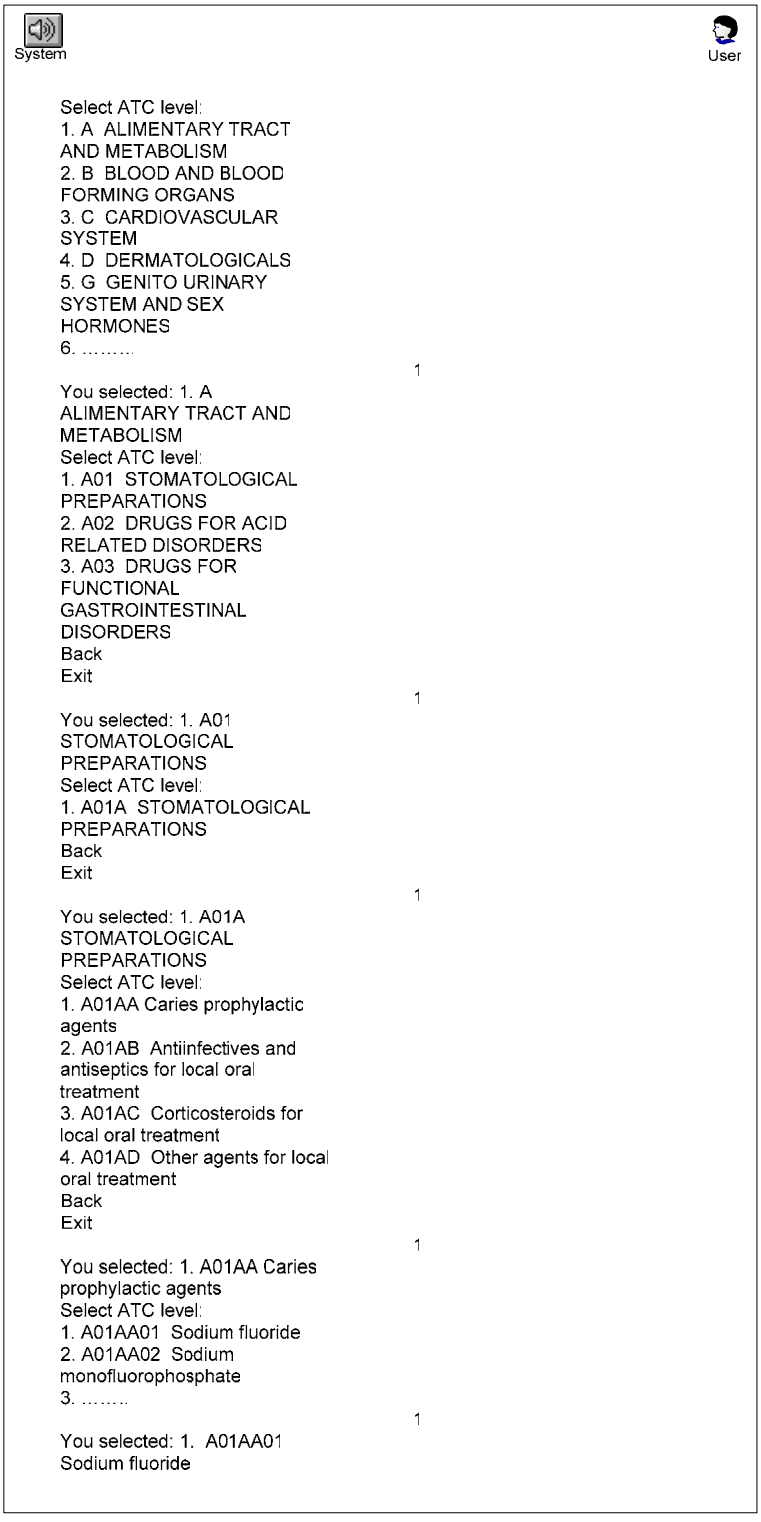

Figure 52: ATC Level Selection interactor

OrdinateView

This class will need the *initialize* method to be defined where the *description* attribute and all the *visible, enable* and *index* attributes for each attribute in the referenced entity class can be initialized. The referenced class in this case is the *Medication* class. (Figure 53) is a description on how the interactor will interact with the user. The generated code can be viewed in Appendix D, page 195.

Stereotype: <<elementview>>

OCL expressions:

- *context medication::ui::phone::OrdinateView def: initialize():Void = Tuple{description = "New #space# Medication", drugIndex = 0, startDateIndex = 1, endDateIndex = 2, dosageIndex = 3, instructionIndex = 4, ordinatorVisible = false, seponatorVisible = false, drugLabel = "Drug:", dosageLabel = "Dosage:", startDateLabel = "Ordination Date:", endDateLabel = "Seponation Date:", instructionLabel = "Instruction:"}*
- *context medication::ui::phone::OrdinateView def: getDrugDisplayName(element: oclAny):String = element.oclAsType(medication::businessEntities::Drug).getCode().concat('#space#').concat(element.oclAsType(medication::businessEntities::Drug).getName())*

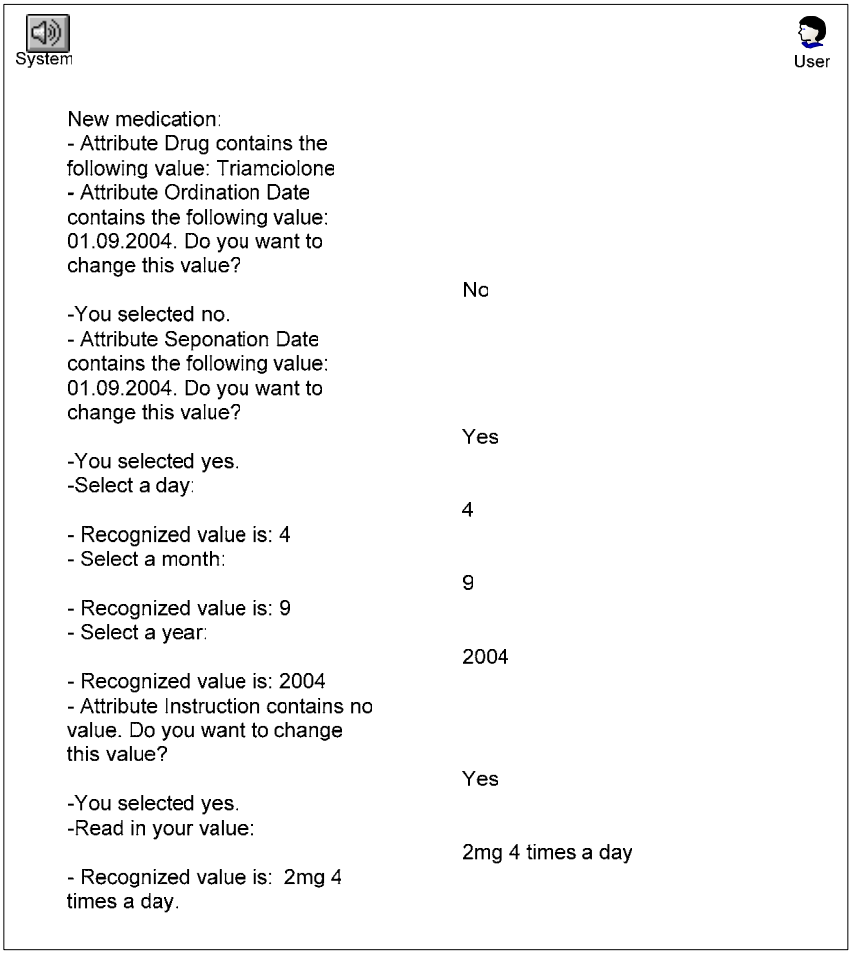

Figure 53: Ordinate View interactor

4.2 Summary

We have in this chapter described a case with demands for supporting mobility in how and where the work is done. The case was the doctor's medication tasks in a patient consultation at the patient's home. The proposed framework from chapter 3 was used to specify and develop the system on two types of platforms: Java enabled mobile phones with portrait mode screens and mobile phone keyboards and a speech based interface that runs on a server which can be called from any phone. The forthcoming chapter will evaluate and discuss our experiences from this chapter.

5. Evaluation and Discussion

Our aim was to introduce a framework for task- and model-based user interface development for multi interface mobile systems. This framework should be based on existing approaches and focus on supporting platform independent development. In this context platform independence means independence of user interface types like graphical user interfaces and speech based user interface, and independence of device type and their characteristics like screen sizes and keyboard types. In this chapter we evaluate the proposed framework relative to our goals presented in chapter 2.

5.1 Evaluation of the Proposed Framework

Our main goal was to propose an approach for platform independent user interface development for mobile systems, but it was also important that was based on existing well established approaches for user interface development. In chapter 2 we presented different approaches with different dimensions to issues on user interface development. These are user centered approaches, designer centered approaches and platform independence.

5.1.1 User centered

Our approach is task-based which has proven to be a user centered approach by ensuring that the system features is compatible with the tasks to be performed. This is done by first specifying the tasks to be performed and then designing the system to support them. [3-5, 11] Chapter 2 includes a set of guidelines to ensure this process and should be used with the framework.

5.1.2 Designer centered

The framework supports the designers by introducing a process, modeling languages and tools for code generation. The modeling languages for task and dialogue models have been well tested in their fields and proven to hold the qualities needed [11]. In the lack of modeling tools supporting TaskMODL and DiaMODL, these models where created using the tool Microsoft Visio [14]. As a consequence of this the framework has some limitations:

- There is no way to let the modeling tool ensure that the dialogue models directly relate to- and are compatible with the task models.
- There is no way to export the models to an appropriate format that can be used by a transformation engine like a code generator. We had to do a workaround by modeling the interactors as UML [26] classes in the platform specific models which could be exported as XMI[20]. And as a further consequence the framework could not automatically generate the flow and layout between the interacotors.

The development period used to develop user interfaces for different platforms is decreased since one abstract model is used and is, with a minimal set of platform specific specifications, automatically realized for many platforms. This is compared to the other approaches introduces in chapter 1 for developing mobile systems that should run on a variety of devices. The models will create the foundation for the system documentation and ease the documentation process.

5.1.3 Platform independence

The framework is model-based and includes a set of patterns at such an abstraction level that they are platform independent in terms of device type and user interface type. We belive that this set of patterns will cover all the operational needs for most information systems. This assumption is based on analyzing existing information systems while evolving the patterns and using them on a case study. However we have found that the abstraction level is too low to exploit the characteristics on each platform in certain situations: A common task in an information system is to view the information in a specific entity in a set. Two ways of handling this is either to:

- Listing all the enitities and all the relevant attributes of the entities.
- Listing all the entities where each entity is identified by presenting a limited set of its attributes. The user can then select an entity and then all relevant attributes of the selected entity is listed.

The former is often the first choice, but can in many situations not be realizable because of limitations on the platform. In our approach this choice will have to be taken while modeling the abstract model and will in some cases restrict the system to exploit the qualities of specific platforms.

A limited number of patterns and platformes are two of the criterias for this approach to succeed [12]. The set of patterns does only consist of five patterns. We use user context analysis to find the platforms to be supported. The analysis task is supported by a set of guidelines. This should help the designers find a limited set of supported platforms and it should be manageble to realize the patterns for each platform. Our patterns have proven to be realizable on two different platforms. This is not a huge number, but they are quite different in nature and we will claim that they therefore are likely to be realizable on a variety of platforms.

Despite our experiences with the patterns, we think they need more maturation and evalutations in more case studies to be sure they cover the most common scenarios in information systems, their platform independence and that they are realizable on most platforms.

5.2 Summary

We have evaluated and discussed our experiences using the proposed framework from chapter 3 as used in the example of chapter 4. We belive that the framework has met our goals and criterias, but has some shortcomings in the abstraction level of the patterns. The framework does also need a better modeling tool support to be able to automatically realize the flow and layout between the interactors and help the designer keep the relations between the models.

6. Conclusion and Further Work

This chapter presents an overall summary and contributions of the thesis before we conclude with areas for further work.

6.1 Concluding Remarks

The trend today is the demand for flexibility of where and how work is done and this require flexibility of how, where, in which situations, and from what types of devices information systems can be accessed. At present there is a plethora of different types of mobile devices supporting different situations and users, but there is a lack of development approaches that support this trend.

We have proposed a framework for multi-interface user interface development for mobile systems. It is based on existing task- and model-based approaches and by this keeps both a user centered and designer centered approach. Model-based theories are also used to solve the demand for platform independence. This is done by creating patterns in such an abstraction level that the platform specific characteristics are avoided as elements in the language for dialogue modeling. These models are then, with a limited set of platform specific parameters, realized to different types of platforms thru code generation.

6.2 Further Work

This section presents topics for further work. A variety of areas are discovered that should receive closer attention. Topics refer both to general user interface development issues and mobile user interface development issues.

6.2.1 Modeling tool for TaskMODL and DiaMODL

A modeling tool should be developed that is specialized for these languages. The tool should support traceability between the the different models and have export functions to formats that a code generator should be able to handle. XMI might be evaluated as such a format. Our code generator will then need extensions to handle the metamodel for DiaMODL.

6.2.2 Patterns in a higher abstraction level

There should be developed sets of patterns in higher abstraction levels to try to extend the framework to be even more user interface type independent like discussed in chapter 5. This should be based on the work started by [12].

6.2.3 Refining and extending the platform specific models and transformation rules for our set of patterns

One of the criterias for the framework is to limit the amount of specifications in the concrete dialogue models. There is room for improvement on this in the existing set of patterns. The domain entities can e.g. be extended with invariants like valid length on strings, valid ranges on numbers, initial values etc. This can be used by the transformation rules to create a more adapted user interface. This work can be based on the theories in [16].

References

1. Nilsson, E.G. *User Interface Modeling and Mobile Applications - Are We Solving Real World Problem?* in *Tamodia '2002*. 2002.
2. Krogstie, J., et al., *Usable mCommerce Systems: The Need for Modeling-Based Approaches.* 2002.
3. Wilson, S. and P. Johnson. *Bridging the Generation Gap: From Work Tasks to User Interface Designs*. in *CADUI96*. 1996.
4. Veer, G.v.d. and M.v. Welie. *Task Based Groupware Design: Putting Theory into Practice*. in *Designing Interactive Systems - DIS 2000*. 2000. New York.
5. Welie, M.v., G.C.v.d. Veer, and A. Eliëns. *An Ontology for Task World Models*. in *DSV-IS'98*. 1998. Wien: Springer-Verlag.
6. Harrison, S. and P. Dourish. *Re-placing space: The Roles of Place and Space in Collaborative Systems*. in *ACM Conference on Computer Supported Cooperative Work (CSCW)*. 1996. Boston, USA: ACM Press, New York.
7. Luff, P. and C. Heath. *Mobility in Collaboration*. in *Computer Supported Collaborative Work Conference (CSCW)*. 1998. Seattle, Washington.
8. Bergquist. *På spaning efter mobilt liv*. in *Killer Applications for the Mobile Society*. 1999.
9. Alexander, C., *The Timeless Way of Building*. 1979: Oxford University Press.
10. Agre, P.E., *Changing Places: Contexts of Awareness in Computing.* Human-Computer Interaction, 2001. **16 (2-4)**: p. 177-192.
11. Trætteberg, H., *Model-based User Interface Design*, in *Department of Computer and Information Sciences*. 2002, Norwegian University of Science and Technology: Trondheim.
12. Nilsson, E.G. *Combining Compound Conceptual User Interface Components with Modelling Patterns - a Promising Direction for Model-based Cross-platform User Interface Development*. in *DSV-IS'02*. 2002.
13. Gamma, E., et al., *Design Patterns - Elements of Reusable Object-Oriented Software*. 1995: Addison-Wesley.
14. Microsoft, *Visio*. 2002.
15. OMG, *MDA Specifications*. 2003, OMG.
16. Janssen, C., A. Weisbecker, and J. Ziegler. *Generating User Interfaces from Data Models and Dialogue Net Specifications*. in *INTERCHI'93*. 1993.
17. Sun, *Java 2 Platform, Micro Edition White Papers*. 2003, Sun Microsystems.
18. Sun, *Java 2 Platform, Standard Edition, White Papers*. 2003, Sun Microsystems.
19. OMG, *UML 2.0 OCL Specification*. 2003, OMG.
20. OMG, *Meta Object Facility (MOF) 2.0 XMI Mapping Specification*. 2003, OMG.
21. ArgoUML, *ArgoUML Documentation*. 2004.
22. Oldevik, J., *UMT: UML Model Transformation Tool*. 2003.
23. WHO, *The ATC/DDD system*. 2004, WHO Collaborating Centre for Drug Statistics Methodology: Oslo.
24. Baar, D.J.M.d., J.D. Foley, and K.E. Mullet. *Coupling Application Design And User Interface Design*. in *CHI'92*. 1992.
25. Cloudgarden, *Talking Java API*. 2004, Cloudgarden.
26. OMG, *UML 2.0 Specification*. 2003, OMG.
27. Trætteberg, H. *Dialog modelling with interactors and UML Statecharts - A hybrid approach*. in *DSVIS-2003*. 2003. Funchal, Madeira.

Appendix A: TaskMODL

We have chosen [11]'s TaskMODL as the task modeling language in our approach. This chapter is a short introduction to the language.

TaskMODL initially used RML for modeling the domain objects, but [27] introduces a hybrid approach using UML state charts for modeling the domain objects in the related language DiaMODL [11, 27]. We have chosen to use this approach for TaskMODL as well. Domain objects are therefore mainly modeled as UML *objects* or UML *multi objects* [26] (Figure 54).

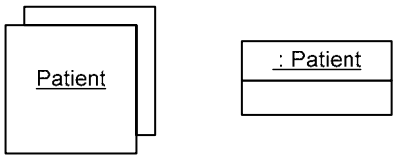

Figure 54: UML objects and UML multi objects

The main construct of the language is the *task* element. Tasks are modeled with a rounded rectangle with a top compartment containing the task's name, an optional middle compartment containing a textual description or decomposition into subtasks and an optional bottom compartment containing resources related to the task. The name should be descriptive and contain a number in a hierarchical style, like "1.1 Sub-task1". The resources can be other constructs like *actors* (a person playing a specific role) and/or *objects* related to the task. It can also have an object or a *multiobject* as input and/or output flow. (Figure 55)

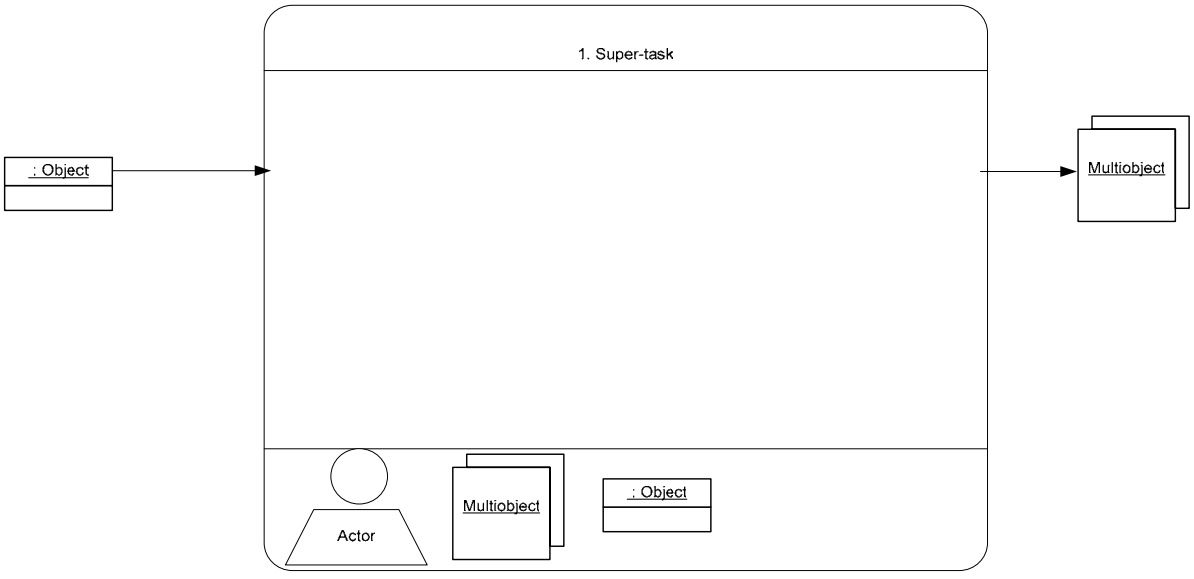

Figure 55: Task with related elements

There are two ways for modeling sequence. The easiest way is using a *flow array* that may contain an object or multi object between tasks (Figure 56). The alternative way is using one of the four sequence constraints in (figure x) modeled as in (Figure 57).

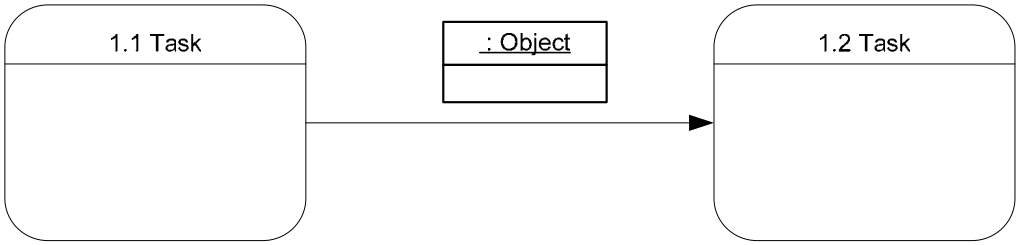

Figure 56: Sequence specification type one

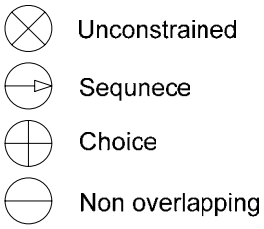

Figure 57: Sequence specification type two

Appendix B: DiaMODL

We have chosen [11, 27]'s TaskMODL as the modeling language for our approach. This chapter is a short introduction to the language.

DiaMODL initially use RML for modeling the domain objects, but [27] introduces a hybrid approach using UML state charts for modeling the domain objects and we have chosen to use this approach.. Domain objects are therefore mainly modeled as UML *objects* or UML *multi objects* [26].

The main concept of the language is *interactors* which is a component that can send and receive information between the user and the system. Information is sent and received through four different types of *gates*:

1. Input/send (is): Input sent from the interactor to the system.
2. Output/receive (or): Output from the system received by the interactor.
3. Input/receive (ir): Input received by the interactor from the user.
4. Output/send (os): Output from the interactor towards the user.

The system dimension is always to the left of the interactor and the user dimension is to the right of the interactor. An interactor can also be a composite of other interactors in sequence. Only gates type 1 and 2 are used when the interactor communicates directly with the user. (Figure 58)

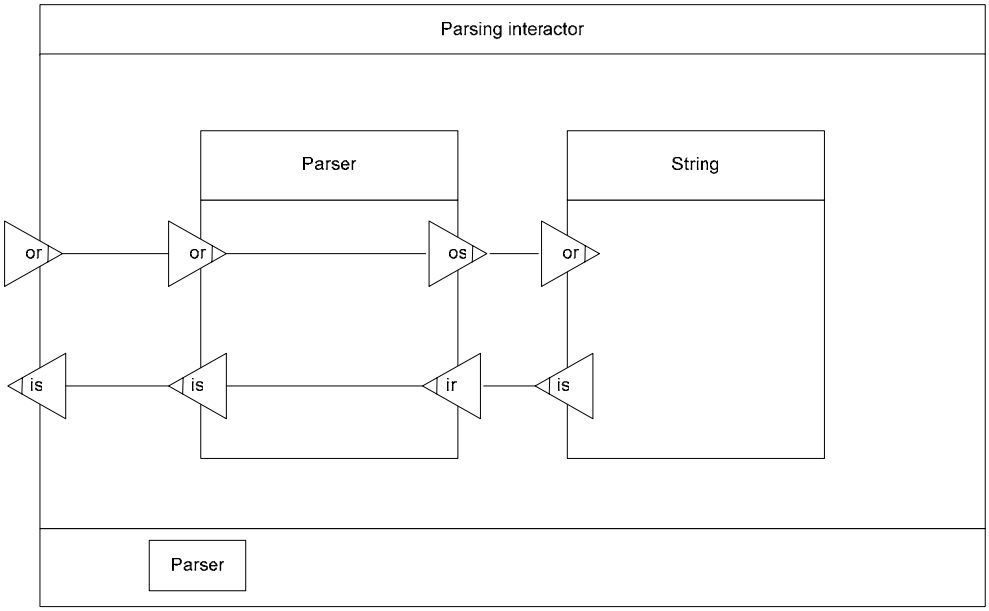

Figure 58: Interactor

Flow arrays can also be used to indicate sequences when this is not obvious by the gates. When an interactor can lead to different states a *state* construct with outgoing flow arrays can be used (Figure 59).

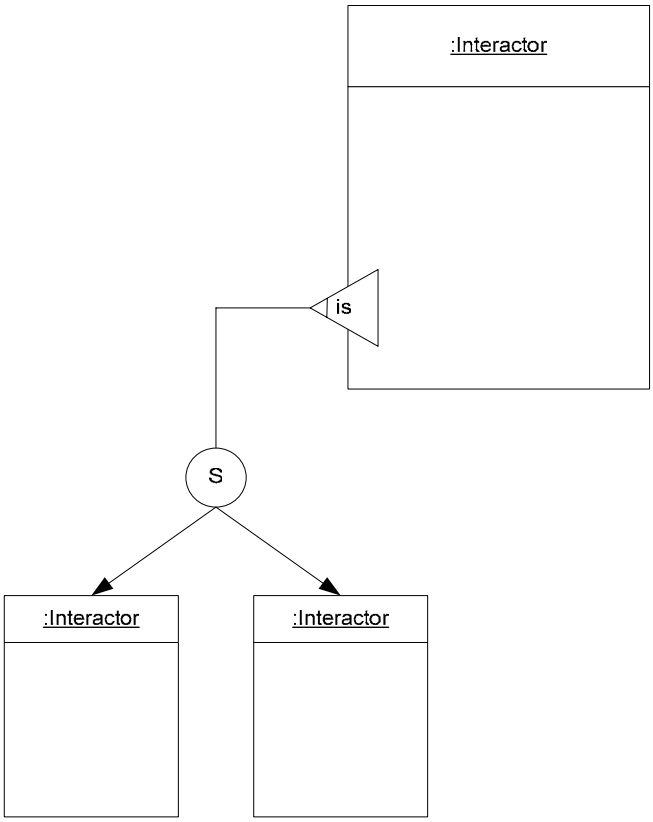

Figure 59: States

Appendix C: UmlAPI - The Code Generator

UmlAPI is a generator tool that can generate code from UML [26], models represented as XMI [20], and a set of transformation rules. The generator can read project files written in xml. These project files have a reference to an XMI file and references from packages to profiles which are sets of transformation rules. An example of Profiles can be a set of transformation rules supporting a specific user interface type or platform. A profile file includes references to different rule files and each reference is marked with a stereotype to match, a UML metamodel element to match and a file extension that together with the element name creates the file name.

Each rule file consists of transformation rules expressed in an xml language. The language consists of the following elements:

- A compulsory <Template> tag which is the root element of the file.
- <Element> tags that have a 'member' attribute which is an operation that is invoked on the active UML element. When a XMI file is loaded, an object structure based on the UML metamodel is built and information can be retrieved using the member attribute value, which is an operation name executed using reflection. Default active UML element is the UML element triggering the rule.
- <Foreach> tags that have an 'expression' attribute. This attribute's value is an operation in the UML structure returning an array of elements. All the child elements of the tag are processed once for each element in the array where the element is set to be the active element.
- <If> tags which have an 'expression' attribute. This attribute's value is an operation in the UML structure returning a Boolean value. A '!' can be placed in the front of the expression to indicate NOT.

Since each expression called towards the UML structure is based on Java Reflection, other general Java expressions, like String manipulations, can be used if they return an expected result.

Most similar tools we have tested are based on transformation rules expressed in XSLT querying an XMI file. Our experience with this type of transformations is that the XSLT files gets quite complex and large This tool and the way of handeling the transformations was also (in addition to handle code generations in the framework) ment as a experiment to see if the transformation rules could be expressed less complex. Our evaluation of this is that the transformation rules ended up with less code and more tidy, but the statements querying the UML model often was very complex to write (this was easier with in XSLT with already well defined templates). A better tool that does automatic syntax-checks and -suggestions may be a solution to improve this. Our experience is also that it is harder to get nice layout in the generated code with this approach compared to the XSLT approach.

Figure 60 shows a subset of the structure of the UML object representation and all operations that can be invoked from the transformation rules.

The generator can also handle one type of OCL 2.0 expressions; operation definition expressions. Some modeling tools like ArgoUML v0.14 [21] do not support OCL 2.0 and therefore not 'def' expressions, and as a workaround all UML tagged values with the tag name 'ocl' will be handled as OCL expressions.

Note that this tool was built to support the framework and case study presented in this report and therefore may not have all features need to support all types of system specifications.

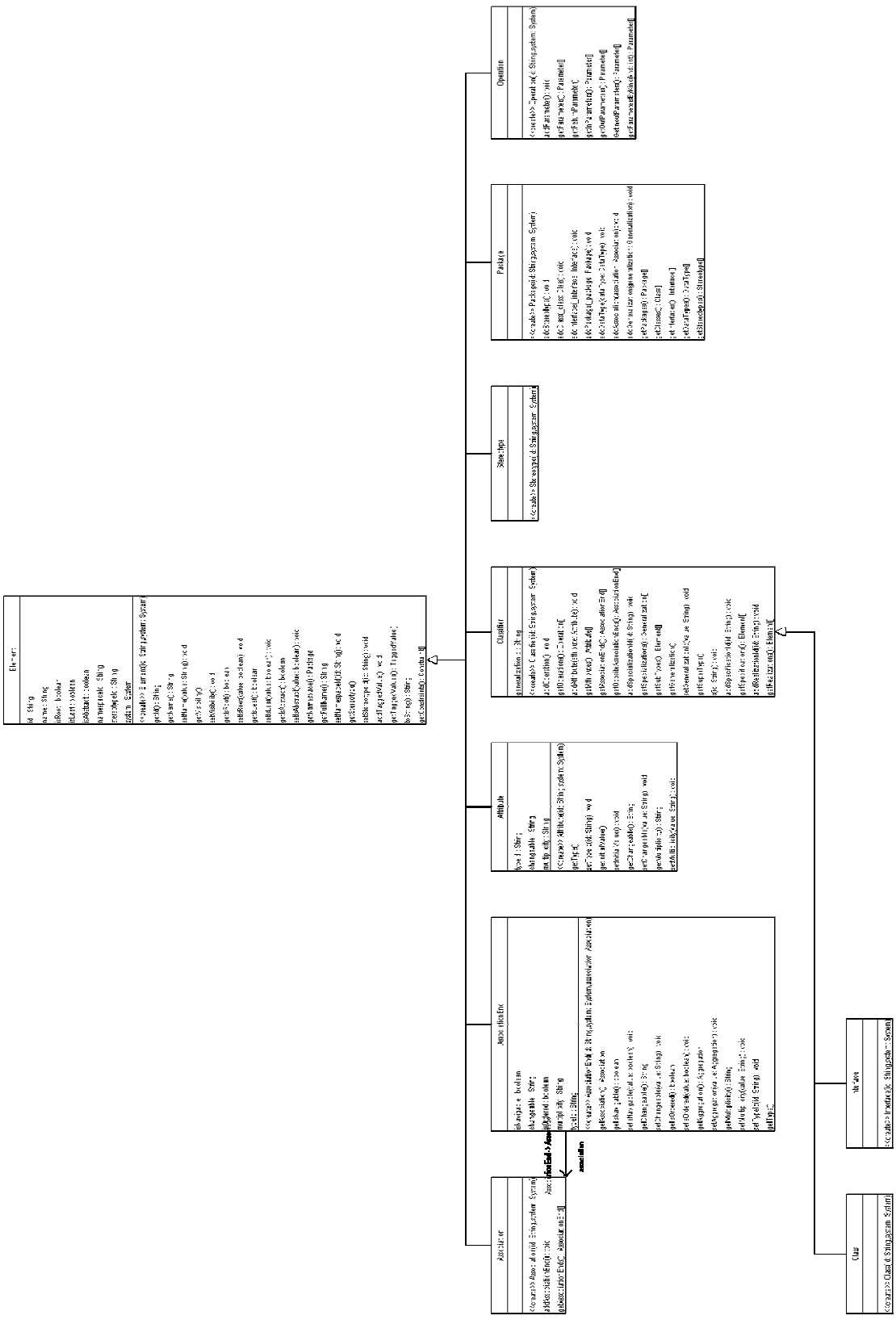

Figure 60: UmlAPI – Subset of the UML metamodel

Appendix D: Source Files

The follwing sections will include source files referenced in the earlier chapters.

The Pattern Source Files – JInteractorControls Library

This section include the source files to the existing components and transformation rules used in the framework.

Platform Independent Source Files

Common Files

These are files not directly related to a pattern, but are used by the pattern files.

JavaEntity.xml

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

import java.lang.*;
import java.util.Vector;
import jinteractorcontrols.Date;

public class <Element member="getName"/> <If
expression="getGeneralization">extends <Identifier
member="getGeneralization.getSuperType.getName"/></If>{

      <Foreach expression="getAttributes">private <Identifier
member="getType.getName"/> <Identifier member="getName"/>;
      </Foreach>
      <Foreach expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('0..*')"><If
expression="getVisibility.getName.startsWith('public')">
      private Vector <Identifier member="getName"/>;</If></If>
      <If expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
      private <Identifier member="getType.getName"/> <Identifier
member="getName"/>;</If></If></Foreach>

      <If expression="!getGeneralization">
      public <Element member="getName"/>(){
        super();
        <Foreach expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('0..*')"><If
expression="getVisibility.getName.startsWith('public')">
        this.<Identifier member="getName"/> = new
Vector();</If></If></Foreach>
      }</If>

      public <Element member="getName"/>(<If
expression="getGeneralization"><If
expression="getGeneralization.getSuperType.getGeneralization">
        <Foreach
expression="getGeneralization.getSuperType.getGeneralization.getSuperType.get
Attributes" listSeparator=","><Identifier member="getType.getName"/>
<Identifier member="getName"/>
        </Foreach>,</If>
```

```
        <Foreach expression="getGeneralization.getSuperType.getAttributes"
listSeparator=","><Identifier member="getType.getName"/> <Identifier
member="getName"/></Foreach></If>
        <Foreach expression="getAttributes" listSeparator=","><Identifier
member="getType.getName"/> <Identifier member="getName"/></Foreach>){
        <If expression="!getGeneralization">
        this();</If>
        <If expression="getGeneralization"><If
expression="getGeneralization.getSuperType.getGeneralization">super(<Foreach
expression="getGeneralization.getSuperType.getGeneralization.getSuperType.get
Attributes" listSeparator=","><Identifier
member="getName"/></Foreach></If><Foreach
expression="getGeneralization.getSuperType.getAttributes"
listSeparator=","><Identifier member="getType.getName"/> <Identifier
member="getName"/></Foreach>);</If>

        <Foreach expression="getAttributes">this.<Identifier
member="getName"/> = <Identifier member="getName"/>;
        </Foreach>
      }

      <Foreach expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('0..*')"><If
expression="getVisibility.getName.startsWith('public')">
      <Identifier member="getVisibility.getName"/> Vector get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
){return this.<Identifier member="getName"/>;}
      <Identifier member="getVisibility.getName"/> void add<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
<Identifier member="getType.getName"/> value){this.<Identifier
member="getName"/>.addElement(value);}
      <Identifier member="getVisibility.getName"/> void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
Vector value){this.<Identifier member="getName"/>=value;}</If></If>
      <If expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
      <Identifier member="getVisibility.getName"/> <Identifier
member="getType.getName"/> get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
){return this.<Identifier member="getName"/>;}
      <Identifier member="getVisibility.getName"/> void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
<Identifier member="getType.getName"/> value){this.<Identifier
member="getName"/>=value;}</If></If></Foreach>

      <Foreach expression="getAttributes">
      <Identifier member="getVisibility.getName"/> <Identifier
member="getType.getName"/> get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
){return this.<Identifier member="getName"/>;}
      <Identifier member="getVisibility.getName"/> void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
<Identifier member="getType.getName"/> value){this.<Identifier
member="getName"/>=value;}</Foreach>

      <Foreach expression="getOperations">
```

```
      <Identifier member="getVisibility.getName"/> <Identifier
member="getReturnParameter.getType.getName"/> <Identifier
member="getName"/>(<Foreach expression="getInParameters"
listSeparator=","><Identifier member="getType.getName"/> <Identifier
member="getName"/></Foreach>);</Foreach>

}

</Template>
```

IActionListener.java

```
package jinteractorcontrols;

/**
 * Interface that must be implemented by callback subscribers.
 */
public interface IActionListener {

  public final static int OK = 0;
  public final static int CANCEL = 1;
  public final static int BACK = 2;
  public final static int EXIT = 3;

      /**
       * Operation implemented by subscribers and called when
       * callbacks are performed.
       * @param source the interactor control performing the callback.
       * @param type the type of callback. Must be one of the static values:
       * IActionListener.OK, IActionListener.CANCEL,
       * IActionListener.BACK or IActionListener.EXIT
       */
      public void actionPerformed(Object source, int type);
}
```

ActionListener.java

```
package jinteractorcontrols;

/**
 * A helper class used to handle callbacks from the interactor controls.
 */
public class ActionListener implements IActionListener {

      public ActionListener() {
            super();
      }

      /*
       * @see IActionListener.actionPerformed(Object source, int type)
       */
      public void actionPerformed(Object source, int type) {
      }
}
```

Date.java

```
package jinteractorcontrols;
```

```
import java.util.Calendar;

/**
 * Handeling dates and date formating.
 * J2ME does not include the same formatter
 * classes as J2SE so this this is a generic
 * class working on both J2ME and J2SE.
 */
public class Date {

  private java.util.Calendar calendar;

  public Date() {
    this.calendar = java.util.Calendar.getInstance();
  }

  /**
   * Constructor with a date time as input.
   * @param date day of month
   * @param month month of year
   * @param year
   * @param hour hour of day
   * @param minute minute of hour
   */
  public Date(int date, int month, int year, int hour, int minute){
    this();
    this.calendar.set(Calendar.DAY_OF_MONTH, date);
    this.calendar.set(Calendar.MONTH, month);
    this.calendar.set(Calendar.YEAR, year);
    this.calendar.set(Calendar.HOUR_OF_DAY, hour);
    this.calendar.set(Calendar.MINUTE, minute);
    this.calendar.set(Calendar.SECOND, 0);
    this.calendar.set(Calendar.MILLISECOND, 0);
  }

  /**
   * Constructor with a date as input.
   * @param date dya of month
   * @param month month of year
   * @param year
   */
  public Date(int date, int month, int year){
    this(date, month, year, 0, 0);
  }


  /**
   * Return this instance as a java.util.Date
   * @return this instance current java.util.Date
   */
  public java.util.Date getDate(){
    return calendar.getTime();
  }

  /**
   * Set the date from a java.util.Date
   * @param date the relevant date
```

```
 */
public void setDate(java.util.Date date){
  this.calendar.setTime(date);
}

/**
 * Set year
 * @param year
 */
public void setYear(int year){
    this.calendar.set(Calendar.YEAR, year);
}

/**
 * Set month
 * @param month month of year
 */
public void setMonth(int month){
    this.calendar.set(Calendar.MONTH, month);
}

/**
 * Set day of month
 * @param day day of month
 */
public void setDayOfMonth(int day){
    this.calendar.set(Calendar.DAY_OF_MONTH, day);
}

/**
 * Set hour
 * @param hour of day
 */
public void setHour(int hour){
    this.calendar.set(Calendar.HOUR_OF_DAY, hour);
}

/**
 * Set minute
 * @param min minute of the hour
 */
public void setMinute(int min){
    this.calendar.set(Calendar.MINUTE, min);
}

/**
 * @see toDateTimeString
 */
public String toString(){
  return toDateTimeString();
}

/**
 * Returns the instance as a date string (dd.mm.yyyy)
 * @return a date string
 */
public String toDateString(){
```

```
    return calendar.get(Calendar.DAY_OF_MONTH) + "." +
calendar.get(Calendar.MONTH) + "." + calendar.get(Calendar.YEAR);
  }

  /**
   * Return the instace as a time string (hh:mm)
   * @return a time string
   */
  public String toTimeString(){
    String hour = "";
    if (calendar.get(Calendar.HOUR_OF_DAY)<10)
      hour += "0";
    hour += Integer.toString(calendar.get(Calendar.HOUR_OF_DAY));
    String minute = "";
    if (calendar.get(Calendar.MINUTE)<10)
      minute += "0";
    minute += Integer.toString(calendar.get(Calendar.MINUTE));
    return hour + ":" + minute;
  }

  /**
   * Returns the instance as a date time string. (dd.mm.yyyy hh:mm)
   * @return a date time string
   */
  public String toDateTimeString(){
    return toDateString() + " " + toTimeString();
  }
}
```

BooleanToInteractor.java

```
package jinteractorcontrols;

/**
 * Platform independent abstract class reflecting
 * OCL::Boolean. It handles all
 * platform independent features related to presenting
 * a OCL::Boolean to the user. All platform
 * specific implementations
 * must inherit this class.
 */
public abstract class BooleanToInteractor {

      protected boolean value;

    /**
     * Consturctor. The initial value is false.
     */
      public BooleanToInteractor() {
            super();
      }

    /**
     * Constructor.
     * @param value the initial value.
     */
```

```
	public BooleanToInteractor(boolean value) {
		this();
		this.value = value;
	}

	/**
	 * Initialize the boolean value.
	 * @param value boolean value.
	 */
	public void setValue(boolean value){
		if (this.value != value){
			this.value = value;
			refresh();
		}
	}

	/**
	 * Refresh the user view. Must be implemented
	 * for each relevant platform.
	 */
	public abstract void refresh();
}
```

ToBooleanInteractor.java

```
package jinteractorcontrols;

/**
 * Platform independent abstract class reflecting
 * OCL::Boolean. It handles all
 * platform independent features related to letting the
 * user send a OCL::Boolean as input to the system.
 * All platform specific implementations
 * must inherit this class.
 */
public abstract class ToBooleanInteractor {

	protected IActionListener listener;
	protected boolean value;

    /**
     * Constructor. Initial value is false.
     */
	public ToBooleanInteractor() {
		super();
		this.value = false;
	}

	/**
	 * Sets a subscriber for the interactor's callbacks.
	 * @param listener a subscriber implementing IActionListener
	 */
	public void setListener(IActionListener listener){
		this.listener = listener;
	}

	/**
```

```
       * Performes the callback.
       * @param type the type of callback. Must be one
       * of the static values:
       * IActionListener.OK, IActionListener.CANCEL,
       * IActionListener.BACK or IActionListener.EXIT
       */
      protected void selectionPerformed(int type){
            if(this.listener != null)
                  this.listener.actionPerformed(this, type);
      }

    /**
     * This operation can be used to get
     * current value.
     * @return the current value
     */
      public boolean getValue(){
            return this.value;
      }
}
```

DateToInteractor.java

```
package jinteractorcontrols;

/**
 * Platform independent abstract class reflecting
 * jinteractorcontrols.Date. It handles all
 * platform independent features related to presenting
 * a jinteractorcontrols.Date to the user. All platform
 * specific implementations
 * must inherit this class.
 */
public abstract class DateToInteractor {

      protected Date value;

      /**
       * Consturctor.
       */
      public DateToInteractor() {
            super();
      }

      /**
       * Constructor.
       * @param value the initial value.
       */
      public DateToInteractor(Date value) {
            this();
            this.value = value;
      }

      /**
       * Initialize the jinteractorcontrols.Date value.
       * @param value jinteractorcontrols.Date value.
```

```
	 */
	public void setValue(Date value){
		if (this.value != value){
			this.value = value;
			refresh();
		}
	}

	/**
	 * Refresh the user view. Must be implemented
	 * for each relevant platform.
	 */
	public abstract void refresh();
}
```

ToDateInteractor.java

```
package jinteractorcontrols;

/**
 * Platform independent abstract class reflecting
 * jinteractorcontrols.Date. It handles all
 * platform independent features related to letting the
 * user send a jinteractorcontrols.Date as input to the system.
 * All platform specific implementations
 * must inherit this class.
 */
public class ToDateInteractor {

	protected IActionListener listener;
	protected Date value;

	/**
	 * Constructor. Initial value is the current date time.
	 */
	public ToDateInteractor() {
		super();
		value = new Date();
	}

	/**
	 * Sets a subscriber for the interactor's callbacks.
	 * @param listener a subscriber implementing IActionListener
	 */
	public void setListener(IActionListener listener){
		this.listener = listener;
	}

	/**
	 * Performes the callback.
	 * @param type the type of callback. Must be one
	 * of the static values:
	 * IActionListener.OK, IActionListener.CANCEL,
	 * IActionListener.BACK or IActionListener.EXIT
	 */
	protected void selectionPerformed(int type){
```

```
            if(this.listener != null)
                  this.listener.actionPerformed(this, type);
      }

      /**
       * This operation can be used to get
       * current value.
       * @return the current value
       */
      public Date getValue(){
            return this.value;
      }
}
```

IntegerToInteractor.java

```
package jinteractorcontrols;

/**
 * Platform independent abstract class reflecting
 * OCL::Integer. It handles all
 * platform independent features related to presenting
 * a OCL::Integer to the user. All platform
 * specific implementations
 * must inherit this class.
 */
public abstract class IntegerToInteractor {

      protected int value;

    /**
     * Constructor
     */
      public IntegerToInteractor() {
            super();
      }

      /**
       * Constructor.
       * @param value the initial value.
       */
      public IntegerToInteractor(int value) {
            this();
            this.value = value;
      }

      /**
       * Initialize the int value.
       * @param value int value.
       */
      public void setValue(int value){
            if (this.value != value){
                  this.value = value;
                  refresh();
            }
      }
```

```
	/**
	 * Refresh the user view. Must be implemented
	 * for each relevant platform.
	 */
	public abstract void refresh();
}
```

ToIntegerInteractor.java

```
package jinteractorcontrols;

/**
 * Platform independent abstract class reflecting
 * OCL::Integer. It handles all
 * platform independent features related to letting the
 * user send a OCL::Integer as input to the system.
 * All platform specific implementations
 * must inherit this class.
 */
public abstract class ToIntegerInteractor {

	protected IActionListener listener;
	protected int value;

	/**
	 * Constructor. Initial value is 0.
	 */
	public ToIntegerInteractor() {
		super();
		this.value = 0;
	}

	/**
	 * Sets a subscriber for the interactor's callbacks.
	 * @param listener a subscriber implementing IActionListener
	 */
	public void setListener(IActionListener listener){
		this.listener = listener;
	}

	/**
	 * Performes the callback.
	 * @param type the type of callback. Must be one
	 * of the static values:
	 * IActionListener.OK, IActionListener.CANCEL,
	 * IActionListener.BACK or IActionListener.EXIT
	 */
	protected void selectionPerformed(int type){
		if(this.listener != null)
			this.listener.actionPerformed(this, type);
	}

	/**
	 * This operation can be used to get
	 * current value.
```

```
	 * @return the current value
	 */
	public int getValue(){
		return this.value;
	}
}
```

StringToInteractor.java

```
package jinteractorcontrols;

/**
 * Platform independent abstract class reflecting
 * OCL::String. It handles all
 * platform independent features related to presenting
 * a OCL::String to the user. All platform
 * specific implementations
 * must inherit this class.
 */
public abstract class StringToInteractor {

	protected String value;

	/**
	 * Consturctor. The initial value is an empty string.
	 */
	public StringToInteractor() {
		super();
		value = new String();
	}

	/**
	 * Constructor.
	 * @param value the initial value.
	 */
	public StringToInteractor(String value) {
		this();
		this.value = value;
	}

	/**
	 * Initialize the String value.
	 * @param value String value.
	 */
	public void setValue(String value){
		if (!this.value.equals(value)){
			this.value = value;
			refresh();
		}
	}

	/**
	 * Refresh the user view. Must be implemented
	 * for each relevant platform.
	 */
	public abstract void refresh();
```

```
}
```

ToStringInteractor.java

```
package jinteractorcontrols;

/**
 * Platform independent abstract class reflecting
 * OCL::String. It handles all
 * platform independent features related to letting the
 * user send a OCL::String as input to the system.
 * All platform specific implementations
 * must inherit this class.
 */
public abstract class ToStringInteractor {

	protected IActionListener listener;
	protected String value;

	/**
	 * Constructor. Initial value is an empty string.
	 */
	public ToStringInteractor() {
		super();
		this.value = "";
	}

	/**
	 * Sets a subscriber for the interactor's callbacks.
	 * @param listener a subscriber implementing IActionListener
	 */
	public void setListener(IActionListener listener){
		this.listener = listener;
	}

	/**
	 * Performes the callback.
	 * @param type the type of callback. Must be one
	 * of the static values:
	 * IActionListener.OK, IActionListener.CANCEL,
	 * IActionListener.BACK or IActionListener.EXIT
	 */
	protected void selectionPerformed(int type){
		if(this.listener != null)
			this.listener.actionPerformed(this, type);
	}

	/**
	 * This operation can be used to get
	 * current value.
	 * @return the current value
	 */
	public String getValue(){
		return this.value;
	}
}
```

The Data Model Transformation Rules (JavaEntity.xml)

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

import java.lang.*;
import java.util.Vector;
import jinteractorcontrols.Date;

public class <Element member="getName"/> <If
expression="getGeneralization">extends <Identifier
member="getGeneralization.getSuperType.getName"/></If>{

      <Foreach expression="getAttributes">private <Identifier
member="getType.getName"/> <Identifier member="getName"/>;
      </Foreach>
      <Foreach expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('0..*')"><If
expression="getVisibility.getName.startsWith('public')">
      private Vector <Identifier member="getName"/>;</If></If>
      <If expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
      private <Identifier member="getType.getName"/> <Identifier
member="getName"/>;</If></If></Foreach>

      <If expression="!getGeneralization">
      public <Element member="getName"/>(){
        super();
        <Foreach expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('0..*')"><If
expression="getVisibility.getName.startsWith('public')">
        this.<Identifier member="getName"/> = new
Vector();</If></If></Foreach>
      }</If>

      public <Element member="getName"/>(<If
expression="getGeneralization"><If
expression="getGeneralization.getSuperType.getGeneralization">
        <Foreach
expression="getGeneralization.getSuperType.getGeneralization.getSuperType.get
Attributes" listSeparator=","><Identifier member="getType.getName"/>
<Identifier member="getName"/>
        </Foreach>,</If>
        <Foreach expression="getGeneralization.getSuperType.getAttributes"
listSeparator=","><Identifier member="getType.getName"/> <Identifier
member="getName"/></Foreach></If>
        <Foreach expression="getAttributes" listSeparator=","><Identifier
member="getType.getName"/> <Identifier member="getName"/></Foreach>){
        <If expression="!getGeneralization">
        this();</If>
        <If expression="getGeneralization"><If
expression="getGeneralization.getSuperType.getGeneralization">super(<Foreach
expression="getGeneralization.getSuperType.getGeneralization.getSuperType.get
Attributes" listSeparator=","><Identifier
member="getName"/></Foreach></If><Foreach
```

```
expression="getGeneralization.getSuperType.getAttributes"
listSeparator=","><Identifier member="getType.getName"/> <Identifier
member="getName"/></Foreach>);</If>

        <Foreach expression="getAttributes">this.<Identifier
member="getName"/> = <Identifier member="getName"/>;
        </Foreach>
      }

      <Foreach expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('0..*')"><If
expression="getVisibility.getName.startsWith('public')">
      <Identifier member="getVisibility.getName"/> Vector get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
){return this.<Identifier member="getName"/>;}
      <Identifier member="getVisibility.getName"/> void add<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
<Identifier member="getType.getName"/> value){this.<Identifier
member="getName"/>.addElement(value);}
      <Identifier member="getVisibility.getName"/> void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
Vector value){this.<Identifier member="getName"/>=value;}</If></If>
      <If expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
      <Identifier member="getVisibility.getName"/> <Identifier
member="getType.getName"/> get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
){return this.<Identifier member="getName"/>;}
      <Identifier member="getVisibility.getName"/> void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
<Identifier member="getType.getName"/> value){this.<Identifier
member="getName"/>=value;}</If></If></Foreach>

      <Foreach expression="getAttributes">
      <Identifier member="getVisibility.getName"/> <Identifier
member="getType.getName"/> get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
){return this.<Identifier member="getName"/>;}
      <Identifier member="getVisibility.getName"/> void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
<Identifier member="getType.getName"/> value){this.<Identifier
member="getName"/>=value;}</Foreach>

      <Foreach expression="getOperations">
      <Identifier member="getVisibility.getName"/> <Identifier
member="getReturnParameter.getType.getName"/> <Identifier
member="getName"/>(<Foreach expression="getInParameters"
listSeparator=","><Identifier member="getType.getName"/> <Identifier
member="getName"/></Foreach>);</Foreach>

}

</Template>
```

The Element Selection Pattern

ElementSelection.java

```
package jinteractorcontrols;

import java.util.Vector;

/**
 * Platform independent abstract class reflecting
 * the ElementSelection pattern. It handles all
 * platform independent features. All platform
 * specific implementations of the pattern
 * must inherit this class.
 */
public abstract class ElementSelection {

	protected Object selectedElement;
	protected Vector elementCollection;
	protected IActionListener listener;
	protected String description;

	public ElementSelection() {
		super();
	}

	  /**
	   * This operation can be used to get
	   * current selected element.
	   * @return the selected element
	   */
	public Object getSelectedElement() {
		return this.selectedElement;
	}

	  /**
	   * Operations that is used to set all
	   * available elements.
	   * @param elements a set of all elements
	   * the user can select between.
	   */
	public void setElements(Vector elements) {
	  this.elementCollection = elements;
	}

	  /**
	   * Sets a subscriber for the interactor's callbacks.
	   * @param listener a subscriber implementing IActionListener
	   */
	public void setListener(IActionListener listener) {
	  this.listener = listener;
	}

	  /**
	   * Performes the callback.
	   * @param type the type of callback. Must be one
	   * of the static values:
```

```
         * IActionListener.OK, IActionListener.CANCEL,
         * IActionListener.BACK or IActionListener.EXIT
         */
      protected void selectionPerformed(int type) {
        if (this.listener != null)
          this.listener.actionPerformed(this, type);
      }

        /**
         * Sets a text describing the interactor control.
         * This can e.g. be an instruction.
         * @param description the text string.
         */
      public void setDescription(String description) {
        this.description = description;
      }

      /**
       * Activates the interactor control and makes
       * it present it self for the user. Must
       * be implemented by all platform specific
       * implementations.
       */
      abstract public void activate();
}
```

The Subset Selection Pattern

SubsetSelection.java

```
package jinteractorcontrols;

import java.util.Vector;

/**
 * Platform independent abstract class reflecting
 * the SubsetSelection pattern. It handles all
 * platform independent features. All platform
 * specific implementations of the pattern
 * must inherit this class.
 */
public abstract class SubsetSelection {

      protected Vector selectedElements;
    protected Vector elementCollection;
    protected IActionListener listener;
    protected String description;

      public SubsetSelection() {
            super();
      }

      /**
       * This operation can be used to get
       * the current selected elements.
       * @return the selected elements
```

```
       */
      public Vector getSelectedElements(){
            return selectedElements;
      }

      /**
       * Operations that is used to set all
       * available elements.
       * @param elements a set of all elements
       * the user can select between.
       */
    public void setElements(Vector elements) {
      this.elementCollection = elements;
    }

      /**
       * Sets a subscriber for the interactor's callbacks.
       * @param listener a subscriber implementing IActionListener
       */
    public void setListener(IActionListener listener) {
      this.listener = listener;
    }

      /**
       * Performes the callback.
       * @param type the type of callback. Must be one
       * of the static values:
       * IActionListener.OK, IActionListener.CANCEL,
       * IActionListener.BACK or IActionListener.EXIT
       */
    protected void selectionPerformed(int type) {
      if (this.listener != null)
        this.listener.actionPerformed(this, type);
    }

      /**
       * Sets a text describing the interactor control.
       * This can e.g. be an instruction.
       * @param description the text string.
       */
    public void setDescription(String description) {
      this.description = description;
    }

      /**
       * Activates the interactor control and makes
       * it present it self for the user. Must
       * be implemented by all platform specific
       * implementations.
       */
    abstract public void activate();
}
```

The Hierarchical Selection Pattern

HierarchicalSelection.java

```
package jinteractorcontrols;

/**
 * Platform independent abstract class reflecting
 * the HierarchicalSelection pattern. It handles all
 * platform independent features. All platform
 * specific implementations of the pattern
 * must inherit this class.
 */
public abstract class HierarchicalSelection {

  protected Object selectedElement;
  protected Object rootElement;
  protected IActionListener listener;
  protected String description;

  public HierarchicalSelection() {
      super();
  }

  /**
   * This operation can be used to get
   * current selected element.
   * @return the selected element
   */
  public Object getSelectedElement() {
          return this.selectedElement;
  }

  /**
   * Operations that is used to set all
   * available elements.
   * @param elements a set of all elements
   * the user can select between.
   */
   public void setRootElement(Object rootElement) {
    this.rootElement = rootElement;
  }

  /**
   * Sets a subscriber for the interactor's callbacks.
   * @param listener a subscriber implementing IActionListener
   */
  public void setListener(IActionListener listener) {
    this.listener = listener;
  }

  /**
   * Performes the callback.
   * @param type the type of callback. Must be one
   * of the static values:
   * IActionListener.OK, IActionListener.CANCEL,
   * IActionListener.BACK or IActionListener.EXIT
   */
  protected void selectionPerformed(int type) {
    if (this.listener != null)
      this.listener.actionPerformed(this, type);
```

```
  }

  /**
   * Sets a text describing the interactor control.
   * This can e.g. be an instruction.
   * @param description the text string.
   */
  public void setDescription(String description) {
    this.description = description;
  }

  /**
   * Activates the interactor control and makes
   * it present it self for the user. Must
   * be implemented by all platform specific
   * implementations.
   */
  abstract public void activate();
}
```

The Interactor Selection Pattern

InteractorSelection.java

```
package jinteractorcontrols;

import java.util.Vector;

/**
 * Platform independent abstract class reflecting
 * the InteractorSelection pattern. It handles all
 * platform independent features. All platform
 * specific implementations of the pattern
 * must inherit this class.
 */
public abstract class InteractorSelection {

      protected Vector commandCollection;
      protected IActionListener listener;
      protected Command selectedCommand;
    protected String description;

      public InteractorSelection() {
            super();
            this.commandCollection = new Vector();
      }

      /**
       * Adds a Command object to the set
       * of available commands.
       * @param cmd an command object
       */
      public void addCommand(Command cmd){
            this.commandCollection.addElement(cmd);
      }
```

```
	/**
	 * This operation can be used to get
	 * current selected Command object.
	 * @return the selected Command object
	 */
	public Command getSelectedCommand(){
		return this.selectedCommand;
	}

	/**
	 * Sets a subscriber for the interactor's callbacks.
	 * @param listener a subscriber implementing IActionListener
	 */
	public  void addListener(IActionListener listener){
		this.listener = listener;
	}

	/**
	 * Performes the callback.
	 * @param type the type of callback. Must be one
	 * of the static values:
	 * IActionListener.OK, IActionListener.CANCEL,
	 * IActionListener.BACK or IActionListener.EXIT
	 */
	protected void selectionPerformed(int type){
		this.listener.actionPerformed(this.selectedCommand, type);
	}

	/**
	 * Sets a text describing the interactor control.
	 * This can e.g. be an instruction.
	 * @param description the text string.
	 */
	public void setDescription(String description) {
	  this.description = description;
	}

	/**
	 * Activates the interactor control and makes
	 * it present it self for the user. Must
	 * be implemented by all platform specific
	 * implementations.
	 */
	abstract public void activate();
}
```

Command.java

```
package jinteractorcontrols;

/**
 * Representing a command in the InteractorSelection pattern.
 */
public class Command {

	protected String name;
```

```
	/**
	 * Constructor
	 * @param name a descriptive name used
	 * to represent the command.
	 */
	public Command(String name) {
		super();
		this.name = name;
	}

	public Command(){
		this("");
	}

	/**
	 * Returns a descriptive name used to
	 * represent the command.
	 * @return the name string
	 */
	public String getName(){
		return this.name;
	}
}
```

The Edit/View Element Pattern

EntityView.java

```
package jinteractorcontrols;

/**
 * Platform independent abstract class reflecting
 * the Edit/View Element pattern. It handles all
 * platform independent features. All platform
 * specific implementations of the pattern
 * must inherit this class.
 */
public abstract class EntityView {

	protected IActionListener listener;
	protected String description;

	public EntityView() {
		super();
	}

	/**
	 * Sets a subscriber for the interactor's callbacks.
	 * @param listener a subscriber implementing IActionListener
	 */
	public void setListener(IActionListener listener){
		this.listener = listener;
	}

	/**
```

```
        * Performes the callback.
        * @param type the type of callback. Must be one
        * of the static values:
        * IActionListener.OK, IActionListener.CANCEL,
        * IActionListener.BACK or IActionListener.EXIT
        */
       protected void selectionPerformed(int type){
             if(this.listener != null)
                   this.listener.actionPerformed(this, type);
       }

       /**
        * Sets a text describing the interactor control.
        * This can e.g. be an instruction.
        * @param description the text string.
        */
       public void setDescription(String description){
             this.description = description;
       }

       /**
        * Activates the interactor control and makes
        * it present it self for the user. Must
        * be implemented by all platform specific
        * implementations.
        */
       abstract public void activate();
}
```

Platform Specific Source Files

Mobile Phone

The following sections include source files for classes and transformation rules related to the mobile phone in portrait mode running a MIDP 2.0 Java Virtual Machine.

Common Files

These are files not directly related to a pattern, but are used by the pattern files.

MIDPBooleanToInteractor.java

```
package jinteractorcontrols.midp;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Form;
import jinteractorcontrols.*;

/**
 * Platform specific abstract class reflecting
 * the OCL::Boolean for the MIDP (in this context
 * a mobile phone in portrait mode) user
 * interface platform. It presents an
 * OCL::Boolean value for the user.
 */
public class MIDPBooleanToInteractor extends BooleanToInteractor{
```

```
 private ChoiceGroup choiceGroup;
 private Form form;
 private String label;

 /**
  * Constructor.
  * @param form the current form.
  * @param label an descriptive label
  */
 public MIDPBooleanToInteractor(Form form, String label) {
   super();
   this.form = form;
   this.value = false;
   this.label = label;

   this.choiceGroup = new ChoiceGroup("", ChoiceGroup.MULTIPLE);
   this.choiceGroup.append(this.label, null);
   setValue(this.value);

   this.form.append(this.choiceGroup);
 }

 /**
  * Constructor.
  * @param form the current form.
  * @param label an descriptive label
  * @param value the initial value
  */
 public MIDPBooleanToInteractor(Form form, String label, boolean value) {
   super(value);
   this.form = form;
   this.value = value;
   this.label = label;

   this.choiceGroup = new ChoiceGroup("", ChoiceGroup.MULTIPLE);
   this.choiceGroup.append(this.label, null);
   setValue(this.value);

   this.form.append(this.choiceGroup);
 }

 /**
 * Initialize the boolean value.
 * @param value boolean value.
 */
public void setValue(boolean value){
   super.setValue(value);
   this.choiceGroup.setSelectedIndex(0, this.value);
 }

 /**
 * Refreshes the interactor. This is handeled
 * automatically on this platform.
 */
```

```
 public void refresh(){
  }
}
```

MIDPToBooleanInteractor.java

```
package jinteractorcontrols.midp;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Form;
import jinteractorcontrols.*;

/**
 * Platform specific abstract class reflecting
 * the OCL::Boolean for the MIDP (in this context
 * a mobile phone in portrait mode) user
 * interface platform. It gets an
 * OCL::Boolean value as input
 * from the user.
 */
public class MIDPToBooleanInteractor extends ToBooleanInteractor{
  private ChoiceGroup choiceGroup;
  private Form form;
  private String label;

  /**
   * Constructor.
   * @param form the current form.
   * @param label an descriptive label
   * @param value the initial value
   */
  public MIDPToBooleanInteractor(Form form, String label, boolean value) {
    super();
    this.form = form;
    this.value = value;
    this.label = label;

    this.choiceGroup = new ChoiceGroup("", ChoiceGroup.MULTIPLE);
    this.choiceGroup.append(this.label, null);

    this.form.append(this.choiceGroup);
  }

  /**
    * This operation can be used to get
    * current value.
    * @return the current value
    */
  public boolean getValue(){
    this.value = this.choiceGroup.isSelected(0);
    return super.getValue();
  }
```

```
  /**
   * Activates the interactor and makes it present
   * it self for the user.  This is handeled
   * automatically on this platform.
   */
  public void activate(){}
}
```

MIDPDateToInteractor.java

```
package jinteractorcontrols.midp;

import jinteractorcontrols.DateToInteractor;
import jinteractorcontrols.Date;
import javax.microedition.lcdui.*;

/**
 * Platform specific abstract class reflecting
 * the jinteractorcontrols.Date for the MIDP (in this context
 * a mobile phone in portrait mode) user
 * interface platform. It presents an
 * jinteractorcontrols.Date value for the user.
 */
public class MIDPDateToInteractor extends DateToInteractor {
  public final static int DATE = 0;
  public final static int TIME = 1;
  public final static int DATE_TIME = 2;

  private TextField dateDisabledField;
  private Form form;
  private int type;
  private String label;

  /**
   * Constructor.
   * @param form the current form.
   * @param label an descriptive label
   * @param value the initial value
   * @param type the date format type (DATE, TIME, DATE_TIME)
   */
  public MIDPDateToInteractor(Form form, String label, Date value, int type) {
    this.form = form;
    this.type = type;
    this.value = value;
    this.label = label;

    this.dateDisabledField = new TextField(this.label, valueToString(), getMaxLength(),
TextField.UNEDITABLE);
    this.form.append(this.dateDisabledField);
  }

  /**
   * Constructor.
```

```
   * @param form the current form.
   * @param label an descriptive label
   * @param value the initial value
   */
  public MIDPDateToInteractor(Form form, String label, Date value) {
    this(form, label, value, DATE_TIME);
  }

  /**
   * Converts the jinteractorcontrols.Date to
   * a string based on the current
   * date format type.
   * @return String the current value as string.
   */
  private String valueToString(){
    String valueAsString = value.toDateTimeString();
    if (type == DATE)
      valueAsString = value.toDateString();
    if (type == TIME)
      valueAsString = value.toTimeString();
    return valueAsString;
  }

  /**
   * Gets the right maximum string length
   * for the current date format type.
   * @return int the current maximum length
   */
  private int getMaxLength(){
    if (this.type == DATE){
      return 32;
    } else if (this.type == TIME){
      return 5;
    } else {
      return 16;
    }
  }

  /**
  * Initialize the Date value.
  * @param value Date value.
  */
  public void setValue(Date value){
    super.setValue(value);
    this.dateDisabledField.setString(valueToString());
  }

  /**
  * Refreshes the interactor. This is handeled
  * automatically on this platform.
  */
  public void refresh(){}
}
```

MIDPToDateInteractor.java

```
package jinteractorcontrols.midp;

import jinteractorcontrols.ToDateInteractor;
import jinteractorcontrols.Date;
import javax.microedition.lcdui.*;

/**
 * Platform specific abstract class reflecting
 * the jinteractorcontrols.Date for the MIDP (in this context
 * a mobile phone in portrait mode) user
 * interface platform. It gets an
 * jinteractorcontrols.Date value as input
 * from the user.
 */
public class MIDPToDateInteractor extends ToDateInteractor{
  public final static int DATE = 0;
  public final static int TIME = 1;
  public final static int DATE_TIME = 2;

  private DateField dateField;
  private Form form;
  private int type;
  private String label;

  /**
   * Constructor.
   * @param form the current form.
   * @param label an descriptive label
   * @param value the initial value
   */
  public MIDPToDateInteractor(Form form, String label, Date value) {
    this.form = form;
    this.type = DATE_TIME;
    this.value = value;
    this.label = label;

    this.dateField = new DateField(this.label, DateField.DATE);
    this.dateField.setDate(this.value.getDate());
    this.form.append(this.dateField);
  }

  /**
    * This operation can be used to get
    * current value.
    * @return the current value
    */
  public Date getValue(){
    this.value.setDate(dateField.getDate());
    return super.getValue();
  }

  /**
```

```
   * Activates the interactor and makes it present
   * it self for the user.  This is handeled
   * automatically on this platform.
   */
  public void activate(){}
}
```

MDIPIntegerToInteractor.java

```
package jinteractorcontrols.midp;

import javax.microedition.lcdui.TextField;
import javax.microedition.lcdui.Form;
import jinteractorcontrols.*;

/**
 * Platform specific abstract class reflecting
 * the OCL::Integer for the MIDP (in this context
 * a mobile phone in portrait mode) user
 * interface platform. It presents an
 * OCL::Integer value for the user.
 */
public class MIDPIntegerToInteractor extends IntegerToInteractor {
  private TextField textField;
  private Form form;
  private String label;

  /**
   * Constructor.
   * @param form the current form.
   * @param label an descriptive label
   * @param value the initial value
   */
  public MIDPIntegerToInteractor(Form form, String label, int value) {
    this.form = form;
    this.value = value;
    this.label = label;

    this.textField = new TextField(this.label, Integer.toString(this.value), 10, TextField.UNEDITABLE);

    form.append(this.textField);
  }

  /**
  * Initialize the int value.
  * @param value int value.
  */
  public void setValue(int value){
    super.setValue(value);
    this.textField.setString(Integer.toString(this.value));
  }

  /**
```

```
* Refreshes the interactor. This is handeled
* automatically on this platform.
*/
public void refresh(){}
}
```

MDIPToIntegerInteractor.java

```
package jinteractorcontrols.midp;

import javax.microedition.lcdui.TextField;
import javax.microedition.lcdui.Form;
import jinteractorcontrols.*;

/**
 * Platform specific abstract class reflecting
 * the OCL::Integer for the MIDP (in this context
 * a mobile phone in portrait mode) user
 * interface platform. It gets an
 * OCL::Integer value as input
 * from the user.
 */
public class MIDPToIntegerInteractor extends ToIntegerInteractor{
  private TextField textField;
  private Form form;
  private String label;

  /**
   * Constructor.
   * @param form the current form.
   * @param label an descriptive label
   * @param value the initial value
   */
  public MIDPToIntegerInteractor(Form form, String label, int value) {
    super();
    this.form = form;
    this.value = value;
    this.label = label;

    this.textField = new TextField(this.label, Integer.toString(this.value), 10, TextField.ANY);

    form.append(this.textField);
  }

  /**
   * This operation can be used to get
   * current value.
   * @return the current value
   */
  public int getValue(){
    this.value = Integer.parseInt(this.textField.getString());
    return super.getValue();
  }
```

```
  /**
   * Activates the interactor and makes it present
   * it self for the user.  This is handeled
   * automatically on this platform.
   */
  public void activate(){}
}
```

MIDPStringToInteractor.java

```
package jinteractorcontrols.midp;

import jinteractorcontrols.*;
import javax.microedition.lcdui.*;

/**
 * Platform specific abstract class reflecting
 * the OCL::String for the MIDP (in this context
 * a mobile phone in portrait mode) user
 * interface platform. It presents an
 * OCL::String value for the user.
 */
public class MIDPStringToInteractor extends StringToInteractor{
  private TextField textField;
  private Form form;
  private int maxLength;
  private String label;

  /**
   * Constructor.
   * @param form the current form.
   * @param label an descriptive label
   * @param value the initial value
   */
  public MIDPStringToInteractor(Form form, String label, String value) {
    this.form = form;
    this.maxLength = maxLength;
    this.value = value;
    this.label = label;

    this.textField = new TextField(this.label, this.value, this.value.length(), TextField.UNEDITABLE);

    this.form.append(this.textField);
  }

  /**
  * Initialize the String value.
  * @param value String value.
  */
  public void setValue(String value){
    super.setValue(value);
    this.textField.setString(this.value);
```

```
  }

  /**
* Refreshes the interactor. This is handeled
* automatically on this platform.
*/
public void refresh(){}
}
```

MIDPToStringInteractor.java

```
package jinteractorcontrols.midp;

import jinteractorcontrols.*;
import javax.microedition.lcdui.*;

/**
 * Platform specific abstract class reflecting
 * the OCL::String for the MIDP (in this context
 * a mobile phone in portrait mode) user
 * interface platform. It gets an
 * OCL::String value as input
 * from the user.
 */
public class MIDPToStringInteractor extends ToStringInteractor {
  private TextField textField;
  private Form form;
  private int maxLength;
  private String label;

  /**
   * Constructor.
   * @param form the current form.
   * @param label an descriptive label
   * @param value the initial value
   * @param maxLength maximum string length
   */
  public MIDPToStringInteractor(Form form, String label, String value, int maxLength) {
    this.form = form;
    this.maxLength = maxLength;
    this.value = value;
    this.label = label;

    this.textField = new TextField(this.label, this.value, this.maxLength, TextField.ANY);

    this.form.append(this.textField);
  }

  /**
   * Constructor.
   * @param form the current form.
   * @param label an descriptive label
   * @param value the initial value
```

```
   */
  public MIDPToStringInteractor(Form form, String label, String value) {
    this(form, label, value, 255);
  }

  /**
    * This operation can be used to get
    * current value.
    * @return the current value
    */
  public String getValue(){
    this.value = this.textField.getString();
    return super.getValue();
  }

  /**
   * Activates the interactor and makes it present
   * it self for the user.  This is handeled
   * automatically on this platform.
   */
  public void activate(){}
}
```

The Element Selection Pattern

Transformation rules (JavaElementSelection.xml)

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class <Element member="getName"/> extends
jinteractorcontrols.midp.MIDPElementSelection {

        public <Element member="getName"/>(Form parentForm, MIDlet midlet){
          super(parentForm, midlet);

          initialize();

        }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>

}
</Template>
```

MIDPElementSelection.java

```
package jinteractorcontrols.midp;
```

```
import jinteractorcontrols.IActionListener;
import jinteractorcontrols.ElementSelection;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.ItemCommandListener;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;
import jinteractorcontrols.Date;

/**
 * Platform specific abstract class reflecting
 * the ElementSelection pattern for the MIDP (mobile
 * in portrait mode) user interface platform.
 * It is a helper class inherited
 * by generated classes using the transformation rules
 * in JavaElementSelection.xml in the speech profile.
 */
public abstract class MIDPElementSelection extends jinteractorcontrols.ElementSelection implements
ItemCommandListener, CommandListener{

  protected int RADIO_BUTTON_MAX = 2;

  protected Form parentForm;
  protected MIDlet midlet;

  protected int formIndex;
  protected int selectionType;
  protected int NO_SET = -1;
  protected int RADIO_BUTTON = 0;
  protected int LIST = 1;
  protected int CALENDAR = 2;
  protected MIDPDateInteractor calendar;

  /**
   * Constructor. Setting the parent from and midlet.
   * @param parentForm
   * @param midlet
   */
  public MIDPElementSelection(Form parentForm, MIDlet midlet) {
    this.parentForm = parentForm;
    this.midlet = midlet;
    this.formIndex = -1;
    this.selectionType = -1;
  }

  /**
   * Hides the interactor from the screen.
   */
```

```
public void hide(){
    if (this.selectionType == RADIO_BUTTON){
        this.parentForm.delete(this.formIndex);
    } else if (this.selectionType == LIST){
        Display.getDisplay(this.midlet).setCurrent(this.parentForm);
    }
}

/**
 * Shows the interactor on the screen.
 */
public void activate() {
  String elementClassName = this.elementCollection.firstElement().getClass().getName();
  if (this.elementCollection.firstElement().getClass().getName().equals("jinteractorcontrols.Date")) {
    activateCalendar();
  }
  else if (this.elementCollection.size() <= RADIO_BUTTON_MAX) {
    activateRadioButtons();
  }
  else {
    activateList();
  }
}

/**
 * Presents the interactor using MIDP radio buttons.
 */
protected void activateRadioButtons(){
    ChoiceGroup group = new ChoiceGroup(this.description, Choice.EXCLUSIVE);
    for (int i = 0; i < this.elementCollection.size(); i++){
        Object element = this.elementCollection.elementAt(i);
        group.append(getDisplayName(element), null);
    }
    group.setItemCommandListener(this);
    this.formIndex = this.parentForm.append(group);
    this.selectionType = RADIO_BUTTON;
    Display.getDisplay(this.midlet).setCurrent(this.parentForm);
}

/**
 * Presents the interactor as a MIDP list
 */
protected void activateList(){
    List list = new List(this.description, List.IMPLICIT);
    list.addCommand(new javax.microedition.lcdui.Command("Back", Command.BACK, 0));
    list.addCommand(new javax.microedition.lcdui.Command("Exit", Command.BACK, 1));

    for (int i = 0; i < this.elementCollection.size(); i++){
        Object element = this.elementCollection.elementAt(i);
        list.append(getDisplayName(element), null);
    }
    list.setCommandListener(this);
    Display.getDisplay(this.midlet).setCurrent(list);
    this.selectionType = LIST;
```

```
        this.formIndex = NO_SET;
  }

  /**
   * Presents the interactor as a MIDP calendar.
   */
  protected void activateCalendar(){
    this.calendar = new MIDPDateInteractor(this.parentForm, "Date: ",
(Date)this.elementCollection.firstElement(), MIDPDateInteractor.DATE, true);

    this.parentForm.addCommand(new javax.microedition.lcdui.Command("Ok", Command.OK, 0));
    this.parentForm.addCommand(new javax.microedition.lcdui.Command("Back", Command.BACK, 1));
    this.parentForm.addCommand(new javax.microedition.lcdui.Command("Exit", Command.BACK, 2));
    this.parentForm.setTitle(this.description);
    this.parentForm.setCommandListener(this);
  }

  /**
   * Returns a string representing an element.
   * Must be implemented by all classes that
   * inherit this class.
   * @param element Object the relevant element.
   * @return String the text representing the element.
   */
  protected abstract String getDisplayName(Object element);

  /**
   * Handeling command selections.
   * @param cmd Command
   * @param disp Displayable
   */
  public void commandAction(Command cmd, Displayable disp){
    if (cmd.getCommandType() == Command.BACK){
      this.selectionPerformed(IActionListener.BACK);
    } else if (cmd.getCommandType() == Command.EXIT){
      this.selectionPerformed(IActionListener.EXIT);
    } else if (cmd.getCommandType() == Command.OK){ //For now this is the way to identify java.util.Date
Selections
      this.selectedElement = this.calendar.getValue();
      this.selectionPerformed(IActionListener.OK);
    } else{
      List list = (List) disp;
      int index = list.getSelectedIndex();
      this.selectedElement = this.elementCollection.elementAt(index);
      Display.getDisplay(this.midlet).setCurrent(this.parentForm);
      this.selectionPerformed(IActionListener.OK);
    }
  }

  /**
   * Handeling direct list selections.
   * @param cmd Command
   * @param item Item
   */
```

```
  public void commandAction(Command cmd, Item item){
     this.selectionPerformed(IActionListener.OK);
  }
}
```

The Hierarchical Selection Pattern

Transformation rules (JavaHierarchicalSelection.xml)

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class <Element member="getName"/> extends
jinteractorcontrols.midp.MIDPHierarchicalSelection {

        public class ESelection extends
jinteractorcontrols.midp.MIDPElementSelection {

          public ESelection(Form parentForm, MIDlet midlet){
            super(parentForm, midlet);
          }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>
        }

        private Form parentForm;
        private MIDlet midlet;


        public <Element member="getName"/>(Form parentForm, MIDlet midlet){
                this.parentForm = parentForm;
                this.midlet = midlet;
                  initialize();
        }

              public void activate() {
                activateSelection((<Foreach
expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')"><Identifier
member="getType.getFullName" /></If></If></Foreach>)this.rootElement);
              }

        public void activateSelection(<Foreach
expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')"><Identifier
member="getType.getFullName" /></If></If></Foreach> element){
          Form form = new Form(this.description);
          ESelection selection = new ESelection(form,
```

```
              this.midlet);

          selection.setElements(element.getChildren());

          selection.setDescription(this.description);
          selection.setListener(new jinteractorcontrols.ActionListener() {
            public void actionPerformed(Object source, int type) {
              atcLevelSelectionPerformed(source, type);
            }
          });

javax.microedition.lcdui.Display.getDisplay(this.midlet).setCurrent(form);
          selection.activate();
        }

        public void atcLevelSelectionPerformed(Object source, int type){
          if (type == jinteractorcontrols.IActionListener.OK){
            jinteractorcontrols.ElementSelection selectionControl =
(jinteractorcontrols.ElementSelection) source;
            <Foreach expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')"><Identifier
member="getType.getFullName" /></If></If></Foreach> element
                  = (<Foreach expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')"><Identifier
member="getType.getFullName" /></If></If></Foreach>)
selectionControl.getSelectedElement();
            if (element.isComposite()) {
              activateSelection(element);
            }
            else {
              this.selectedElement = element;

this.selectionPerformed(jinteractorcontrols.IActionListener.OK);
            }
          }
          else if (type == jinteractorcontrols.IActionListener.BACK) {

this.selectionPerformed(jinteractorcontrols.IActionListener.BACK);
          }
          else if (type == jinteractorcontrols.IActionListener.CANCEL) {

this.selectionPerformed(jinteractorcontrols.IActionListener.CANCEL);
          }
          else if (type == jinteractorcontrols.IActionListener.EXIT) {

this.selectionPerformed(jinteractorcontrols.IActionListener.EXIT);
          }
        }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>
}
</Template>
```

MIDPHierarchicalSelection.java

```
package jinteractorcontrols.midp;

/**
 * Platform specific abstract class reflecting
 * the HierarchicalSelection pattern for the MIDP (mobile
 * in portrait mode) user interface platform.
 * It is a helper class inherited
 * by generated classes using the transformation rules
 * in JavaHierarchicalSelection.xml in the speech profile.
 */
public abstract class MIDPHierarchicalSelection extends jinteractorcontrols.HierarchicalSelection {

  /**
   * Constructor
   */
  public MIDPHierarchicalSelection() {
    super();
  }

  /**
   * Shows the interactor on the screen.
   * Must be implemented by all classes
   * that inherit this class.
   */
  public abstract void activate();
}
```

The Interactor Selection Pattern

Transformation rules (JavaInteractorSelection.xml and JavaCommand.xml)

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class <Element member="getName"/> extends
jinteractorcontrols.midp.MIDPInteractorSelection {


        public <Element member="getName"/>(Form parentForm){
          super(parentForm);

          initialize();

        }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>
```

```
}
</Template>

<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

public class <Element member="getName"/> extends jinteractorcontrols.Command
{

        public <Element member="getName"/>(){
          initialize();
        }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>

}
</Template>
```

MIDPInteractorSelection.java

```
package jinteractorcontrols.midp;

import java.util.Hashtable;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import jinteractorcontrols.IActionListener;
import jinteractorcontrols.InteractorSelection;

/**
 * Platform specific abstract class reflecting
 * the InteractorSelection pattern for the MIDP (mobile
 * in portrait mode) user interface platform.
 * It is a helper class inherited
 * by generated classes using the transformation rules
 * in JavaInteractorSelection.xml in the speech profile.
 */
public class MIDPInteractorSelection extends InteractorSelection implements CommandListener{

        private Form parentForm;
        private Hashtable lcduiCommands;

      /**
        * Constructor. Setting the parent from.
        * @param parentForm
        */
        public MIDPInteractorSelection(Form parentForm){
                super();
                this.parentForm = parentForm;
                this.lcduiCommands = new Hashtable();
          javax.microedition.lcdui.Command lcduiCmd = new javax.microedition.lcdui.Command("Back",
```

```
Command.BACK, 0);
            this.parentForm.addCommand(lcduiCmd);
            lcduiCmd = new javax.microedition.lcdui.Command("Exit", Command.BACK, 1);
            this.parentForm.addCommand(lcduiCmd);
        }

    /**
     * Adds a Command to the set of alternative commands.
     * @param cmd Command
     */
    public void addCommand(jinteractorcontrols.Command cmd){
                super.addCommand(cmd);
                javax.microedition.lcdui.Command lcduiCmd = new
javax.microedition.lcdui.Command(cmd.getName(), Command.SCREEN, lcduiCommands.size()+2);
                this.lcduiCommands.put(cmd.getName(), lcduiCmd);
                this.parentForm.addCommand(lcduiCmd);
        }

    /**
     * Adds a subscriber for callbacks.
     * @param listener IActionListener
     */
    public  void addListener(IActionListener listener){
                super.addListener(listener);
                this.parentForm.setCommandListener(this);
        }

    /**
     * Handles command selections and performes callbacks to
     * the subscribers.
     * @param lcduiCmd Command
     * @param displayable Displayable
     */
    public void commandAction(javax.microedition.lcdui.Command lcduiCmd, Displayable displayable){
      if (lcduiCmd.getCommandType() == javax.microedition.lcdui.Command.SCREEN){
        for (int i = 0; i < this.commandCollection.size(); i++) {
          jinteractorcontrols.Command cmd = (jinteractorcontrols.Command)this.
              commandCollection.elementAt(i);
          if (lcduiCmd.getLabel().trim().equals(cmd.getName().trim())) {
            this.selectedCommand = cmd;
            break;
          }
        }
        this.selectionPerformed(IActionListener.OK);
      }else if (lcduiCmd.getCommandType() == javax.microedition.lcdui.Command.BACK){
        this.selectionPerformed(IActionListener.BACK);
      }else if (lcduiCmd.getCommandType() == javax.microedition.lcdui.Command.EXIT){
        this.selectionPerformed(IActionListener.EXIT);
      }
        }

    /**
     * Shows the interactor on the screen.
     * Does nothing because the commands are
```

```
     * added to an existing screen with other
     * interactors.
     */
    public void activate() {}
}
```

The Edit/View Element Pattern

Transformation rules (JavaElementView.xml)

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class <Element member="getName"/> extends
jinteractorcontrols.EntityView implements CommandListener {

            private MIDlet midlet;
            private Form form;

            private jinteractorcontrols.IActionListener listener;

        <Foreach expression="getOpositeAssociationEnds">
            private <Identifier member="getType.getFullName"/> element;
            </Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private boolean <Identifier member="getName"/>Visible = true;
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private boolean <Identifier member="getName"/>Enabled = true;
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private int <Identifier member="getName"/>Index = 0;
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private String <Identifier member="getName"/>Label = "<Identifier
member="getName"/>";
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private jinteractorcontrols.midp.MIDP<Identifier
member="getType.getName"/>ToInteractor <Identifier
member="getName"/>InteractorR;
```

```
            private jinteractorcontrols.midp.MIDPTo<Identifier
member="getType.getName"/>Interactor <Identifier
member="getName"/>InteractorW;
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private boolean <Identifier member="getName"/>Visible = true;
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private boolean <Identifier member="getName"/>Enabled = true;
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private int <Identifier member="getName"/>Index = 0;
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private String <Identifier member="getName"/>Label = "<Identifier
member="getName"/>";
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private jinteractorcontrols.midp.MIDPStringInteractor <Identifier
member="getName"/>Interactor;
            </If></If></Foreach></Foreach>


        public <Element member="getName"/>(MIDlet midlet) {
          this.midlet = midlet;
          initialize();
          this.form = new Form(this.description);

          this.form.addCommand(new javax.microedition.lcdui.Command("Back",
javax.microedition.lcdui.Command.BACK, 0));
            this.form.addCommand(new javax.microedition.lcdui.Command("Ok",
javax.microedition.lcdui.Command.OK, 0));
            this.form.addCommand(new
javax.microedition.lcdui.Command("Cancel",
javax.microedition.lcdui.Command.CANCEL, 1));
            this.form.setCommandListener(this);
        }
```

```
        <Foreach expression="getOpositeAssociationEnds">
            public void set<Identifier member="getType.getName"/>(<Identifier
member="getType.getFullName"/> value) {this.element = value;}
            </Foreach>

        <Foreach expression="getOpositeAssociationEnds">
            public <Identifier member="getType.getFullName"/> get<Identifier
member="getType.getName"/>() { updateElement(); return this.element;}
            </Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>V
isible(boolean value) { this.<Identifier member="getName"/>Visible = value; }
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>E
nabled(boolean value) { this.<Identifier member="getName"/>Enabled = value; }
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>I
ndex(int value) { this.<Identifier member="getName"/>Index = value; }
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>L
abel(String value) { this.<Identifier member="getName"/>Label = value; }
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>V
isible(boolean value) { this.<Identifier member="getName"/>Visible = value; }
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>E
nabled(boolean value) { this.<Identifier member="getName"/>Enabled = value; }
            </If></If></Foreach></Foreach>
```

```
            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>I
ndex(int value) { this.<Identifier member="getName"/>Index = value; }
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>L
abel(String value) { this.<Identifier member="getName"/>Label = value; }
            </If></If></Foreach></Foreach>

            private void createInteractors(int index){
            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
              if (<Identifier member="getName"/>Visible &amp;&amp;
<Identifier member="getName"/>Index == index){
                if (<Identifier member="getName"/>Enabled)
                  <Identifier member="getName"/>InteractorW = new
jinteractorcontrols.midp.MIDPTo<Identifier
member="getType.getName"/>Interactor(this.form, this.<Identifier
member="getName"/>Label, this.element.get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
)) ;
                  else
                  <Identifier member="getName"/>InteractorR = new
jinteractorcontrols.midp.MIDP<Identifier
member="getType.getName"/>ToInteractor(this.form, this.<Identifier
member="getName"/>Label, this.element.get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
)) ;
              }
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
              if (<Identifier member="getName"/>Visible &amp;&amp;
<Identifier member="getName"/>Index == index)
                <Identifier member="getName"/>Interactor = new
jinteractorcontrols.midp.MIDPStringInteractor(this.form, this.<Identifier
member="getName"/>Label, get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>D
isplayName((Object)this.element.get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
)), 150, false) ;
            </If></If></Foreach></Foreach>

            }
```

```
            private void updateElement(){
            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
              if (<Identifier member="getName"/>Visible &amp;&amp;
<Identifier member="getName"/>Enabled)
                element.set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
<Identifier member="getName"/>InteractorW.getValue());
            </Foreach></Foreach>
            }

              public void activate(){
                  for (int i = 1; i &lt; <Foreach
expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">1+</Foreach></Foreach><Foreach
expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">1+</If></If></Foreach
></Foreach>1; i++)
                        createInteractors(i-1);
              Display.getDisplay(this.midlet).setCurrent(this.form);
            }

            public void setListener(jinteractorcontrols.IActionListener
listener){
              this.listener = listener;
            }

            public void commandAction(javax.microedition.lcdui.Command
lcduiCmd, Displayable displayable){
              if (lcduiCmd.getCommandType() ==
javax.microedition.lcdui.Command.BACK){
                listener.actionPerformed(this,
jinteractorcontrols.IActionListener.BACK);
              }
              if (lcduiCmd.getCommandType() ==
javax.microedition.lcdui.Command.CANCEL){
                listener.actionPerformed(this,
jinteractorcontrols.IActionListener.CANCEL);
              }
              else
                listener.actionPerformed(this,
jinteractorcontrols.IActionListener.OK);
            }



        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>

}
</Template>
```

Speech User Interface

The following sections include source files for classes and transformation rules related to the speech user interface.

Common Files

This include source files not directly related to a pattern, but are used by the pattern source files.

SpeechController.java

```
package jinteractorcontrols.speech;

import java.util.Locale;
import javax.speech.Central;
import javax.speech.EngineList;
import javax.speech.EngineModeDesc;
import javax.speech.recognition.Recognizer;
import javax.speech.recognition.ResultListener;
import javax.speech.recognition.Rule;
import javax.speech.recognition.RuleGrammar;
import javax.speech.synthesis.SpeakableListener;
import javax.speech.synthesis.Synthesizer;
import javax.speech.synthesis.SynthesizerModeDesc;
import javax.speech.synthesis.SynthesizerProperties;
import javax.speech.synthesis.Voice;

/**
 * Handles common input and output
 * speech features.
 */
public class SpeechController {

	public Synthesizer synth;
	public Recognizer rec;
	private SpeakableListener spList;
	private RuleGrammar gram;

	/**
	 * Constructor. Initializes the synthesizer and recognizer.
	 */
	public SpeechController() {
		super();
		this.synth = initSynthesizer();
		this.rec = initRecognizer();
	}

	/**
	 * Dispose operation. Disposes the syntheziser and recognizer.
	 */
	public void dispose(){
		disposeSynthesizer();
		disposeRecognizer();
	}

	/**
	 * Disposes the synthersizer.
	 */
```

```
private void disposeSynthesizer(){
      if (synth != null){
            try {
                  synth.deallocate();
                  synth.waitEngineState(Synthesizer.DEALLOCATED);
            } catch(Exception e2) {e2.printStackTrace(System.out);}
      }
}

/**
 * Disposes the recognizer.
 */
private void disposeRecognizer(){
      if (rec != null){
            try {
                  rec.forceFinalize(true);
                  rec.deallocate();
                  rec.waitEngineState(Recognizer.DEALLOCATED);
            } catch(Exception e2) {e2.printStackTrace(System.out);}
      }
}

/**
 * Initializies and returns a recogniser.
 * @return an initialized recognizer.
 */
private Recognizer initRecognizer(){
      Recognizer r = null;
      try{
            // Create a recognizer that supports English.
            Locale.setDefault(Locale.ENGLISH);
            r = Central.createRecognizer(new EngineModeDesc(null));

            // Start up the recognizer
            r.allocate();
            r.waitEngineState(Recognizer.ALLOCATED);

            this.gram = r.newRuleGrammar("jinteractioncontrols");

      }catch (Exception e){
            e.printStackTrace();
      }

      return r;
}

/**
 * Adds a listener that will recieve callbacks when
 * speech input is recognized.
 * @param resultListener
 */
public void addResultListener(ResultListener resultListener){
      // Add the listener to get results
      rec.addResultListener(resultListener);
}

/**
```

```
	 * Removes a callback listener from the set of listeners.
	 * @param resultListener
	 */
	public void removeResultListener(ResultListener resultListener){
		rec.removeResultListener(resultListener);
	}

	/**
	 * @see public void addRule(String name, String grammar, boolean
isPublic)
	 */
	public void addRule(String name, String grammar){
		addRule(name, grammar, true);
	}


	/**
	 * Adds a recognition rule.
	 * (See Java Speech API documentation)
	 * @param name name of rule.
	 * @param grammar the rule string.
	 * @param isPublic is the grammar public or is it private.
	 */
	public void addRule(String name, String grammar, boolean isPublic){
		try{
			Rule rule = gram.getRule(name);
			if (rule != null){
				gram.deleteRule(name);
			}

			rule = gram.ruleForJSGF(grammar);
			gram.setRule(name, rule, isPublic);

			gram.setEnabled(true);

			// Commit the grammar
			rec.commitChanges();
			rec.waitEngineState(Recognizer.LISTENING);
		}catch (Exception e){
			e.printStackTrace();
		}
	}

	/**
	 * Removes a rule by name.
	 * @param name rule name
	 */
	public void removeRule(String name){
		this.gram.deleteRule(name);
	}

	/**
	 * Puts the recognizer in listening mode.
	 */
	public void startListening(){
		try{
			this.rec.requestFocus();
```

```
                this.rec.waitEngineState(Recognizer.FOCUS_ON);
                this.rec.resume();
                this.rec.waitEngineState(Recognizer.RESUMED);
          } catch(Exception e) {
          e.printStackTrace(System.out);
          }
    }

    /**
     * Makes the recognizer stop listening.
     * @param resultListener
     */
    public void stopListening(ResultListener resultListener){
          try{
                this.rec.releaseFocus();
                this.rec.waitEngineState(Recognizer.FOCUS_OFF);
                this.rec.pause();
                this.rec.waitEngineState(Recognizer.PAUSED);
                this.rec.suspend();
                this.rec.waitEngineState(Recognizer.SUSPENDED);
          } catch(Exception e) {
          e.printStackTrace(System.out);
          }
          this.removeResultListener(resultListener);
    }

    /**
     * Initializes the synthesizer.
     * @return the initialized synthesizer.
     */
    private Synthesizer initSynthesizer(){
          try {
                EngineList list = Central.availableSynthesizers(null);
                SynthesizerModeDesc desc =
(SynthesizerModeDesc)list.elementAt(0); // TODO Dynamic SynthesizerModeDesc
selection.

                Synthesizer synth = Central.createSynthesizer(desc);
                ((com.cloudgarden.speech.CGEngineProperties)

    synth.getSynthesizerProperties()).setEventsInNewThread(false);
                synth.addEngineListener(new TestEngineListener());

                synth.allocate();
                synth.resume();
                synth.waitEngineState(Synthesizer.ALLOCATED);

                Voice[] vs = desc.getVoices();
                Voice v = vs[0]; // TODO Dynamic Voice selection.

                System.out.println("Using voice "+v);
                SynthesizerProperties props =
synth.getSynthesizerProperties();
                props.setVoice(v);
                props.setVolume(1.0f);
                props.setSpeakingRate(200.0f);
                return synth;
```

```
            } catch(Exception e) {
                  e.printStackTrace(System.out);
            } catch(Error e1) {
                  e1.printStackTrace(System.out);
            }
            return synth;
      }

      /**
       * Presents a string for the user using
       * the speech synthesizer.
       * @param arg0 the text to be presented.
       */
      public void speak(String arg0){
            try{
                  this.synth.speak(arg0, spList);
                  this.synth.waitEngineState(Synthesizer.QUEUE_EMPTY);
            } catch(Exception e){
                  e.printStackTrace(System.out);
            }
      }

}
```

SpeechRecognizer.java

```
package jinteractorcontrols.speech;

import java.util.Date;
import javax.speech.recognition.Recognizer;
import javax.speech.recognition.Result;
import javax.speech.recognition.ResultAdapter;
import javax.speech.recognition.ResultEvent;
import javax.speech.recognition.ResultToken;

/**
 * Handles speech recognizing.
 */
public class SpeechRecognizer extends ResultAdapter {
      private Recognizer rec;
      private ISpeechResultListener callbackClass;
      private int objectId;

      /**
       * Constructor.
       * @param rec javax.speech.recognition.Recognizer to be used.
       * @param callbackClass a callback class handeling recognized speech.
       * @param objectId an id representing this initialized object.
       */
      public SpeechRecognizer(Recognizer rec, ISpeechResultListener
callbackClass, int objectId) {
            super();
            this.rec = rec;
            this.callbackClass = callbackClass;
            this.objectId = objectId;
      }
```

```
/**
 * Recieves speech results and performes
 * callback to the listening subscribers.
 */
public void resultAccepted(ResultEvent e) {
      String s = "";
      try{
            Result r = (Result)(e.getSource());
            System.out.print("Result Accepted: "+r);
            ResultToken tokens[] = r.getBestTokens();

            for (int i = 0; i < tokens.length; i++)
                  s += tokens[i].getSpokenText() + " ";

            r.removeResultListener(this);
            rec.releaseFocus();
            rec.pause();
      }
      catch(Exception ex){
            ex.printStackTrace();
      }
      this.callbackClass.resultPerformed(s, this.rec, this.objectId);
}

/**
 * Logs rejected results.
 */
public void resultRejected(ResultEvent e) {
      Result r = (Result)(e.getSource());
      System.out.println("Result Rejected "+r);
}

/**
 * Logs created results.
 */
public void resultCreated(ResultEvent e) {
      Result r = (Result)(e.getSource());
      System.out.println("Result Created ");
}

/**
 * Logs updated results.
 */
public void resultUpdated(ResultEvent e) {
      Result r = (Result)(e.getSource());
      System.out.println("Result Updated... "+r);
      ResultToken[] tokens = r.getBestTokens();
      if(tokens != null && tokens.length > 0) {
            displayTimes(tokens[0]);
      }
}

/**
 * Logs the elapsed result time.
 * @param token the result token.
 */
private void displayTimes(ResultToken token) {
```

```
            Date start = new Date(token.getStartTime());
            Date now = new Date(System.currentTimeMillis());
            System.out.println("Result start =
"+start.getMinutes()+":"+start.getSeconds()+
            ", length = "+((token.getEndTime() -
token.getStartTime())/1000.0)+
            ", now="+now.getMinutes()+":"+now.getSeconds());
      }
}
```

ISpeechResultListener.java

```
package jinteractorcontrols.speech;

import javax.speech.recognition.Recognizer;

/**
 * Interface that must be implemented by callback subscribers.
 */
public interface ISpeechResultListener {

      /**
       * Operation implemented by subscribers and called when
       * callbacks are performed.
       * @param recognizedText the text recognized by the recognizer
       * @param rec the recognizer object.
       * @param objectId the id of the recognizer object.
       */
      public void resultPerformed(String recognizedText, Recognizer rec, int
objectId);
}
```

SpeechResultListener.java

```
package jinteractorcontrols.speech;

import javax.speech.recognition.Recognizer;

/**
 * A helper class used to handle callbacks from the speech recognizer.
 */
public class SpeechResultListener implements ISpeechResultListener {

      public SpeechResultListener() {
            super();
      }

      public void resultPerformed(String recognizedText, Recognizer rec, int
objectId){}
}
```

BooleanToSpeechInterctor.java

```
package jinteractorcontrols.speech;
```

```
import jinteractorcontrols.BooleanToInteractor;

/**
 * Platform specific abstract class reflecting
 * the OCL::Boolean for the speech user
 * interface platform. It presents an
 * OCL::Boolean value for the user.
 */
public class BooleanToSpeechInteractor extends BooleanToInteractor{

      private StringToSpeechInteractor stringInteractor;
      private SpeechController speechController;

      /**
       * Constructor. Initializing the SpeechController.
       * @param speechController
       */
      public BooleanToSpeechInteractor(SpeechController speechController) {
            super();
            this.speechController = speechController;
            stringInteractor = new
StringToSpeechInteractor(this.speechController);
      }

      /**
       * Constructor. Initializing the SpeechController and initial value.
       * @param speechController
       * @param value an initial boolean value.
       */
      public BooleanToSpeechInteractor(SpeechController speechController,
boolean value) {
            super(value);
            this.speechController = speechController;
            stringInteractor = new
StringToSpeechInteractor(this.speechController,
Boolean.toString(this.value));
      }

      /**
       * Re-reads the value.
       */
      public void refresh(){
            this.stringInteractor.setValue(Boolean.toString(this.value));
      }
}
```

ToBooleanInteractor.java

```
package jinteractorcontrols.speech;

import javax.speech.recognition.Recognizer;
import jinteractorcontrols.IActionListener;

/**
 * Platform specific abstract class reflecting
 * the OCL::Boolean for the speech user
```

```
 * interface platform. It gets an
 * OCL::Boolean value as input
 * from the user.
 */
public class ToBooleanInteractor extends
jinteractorcontrols.ToBooleanInteractor implements ISpeechResultListener {

	private SpeechController speechController;
	private SpeechRecognizer speechRecognizer;

	private String ruleName = "boolean";
	private String alternativesGrammar = "yes|no|check|uncheck|checked|not
checked";
	private String displayText = "Select yes or no.";
	protected IActionListener listener;

	/**
	 * Constructor. Initializing the SpeechController.
	 * @param speechController
	 */
	public ToBooleanInteractor(SpeechController speechController) {
		super();
		this.speechController = speechController;
		this.value = false;
	}

	/**
	 * Activates the interactor and makes it present
	 * it self for the user.
	 */
	public void activate(){
		StringToSpeechInteractor displayTextInteractor = new
StringToSpeechInteractor(this.speechController, displayText);
		startSpeechRecognizing();
	}

	/**
	 * Puts the interactor in listening mode
	 * and waits for the user to make a selection.
	 */
	private void startSpeechRecognizing(){
		try{
			this.speechController.addRule(ruleName,
alternativesGrammar);
			if (this.speechRecognizer == null)
				this.speechRecognizer = new
SpeechRecognizer(this.speechController.rec, this, this.hashCode());

	this.speechController.addResultListener(this.speechRecognizer);
			this.speechController.startListening();
		}
		catch (Exception e){
			e.printStackTrace();
		}
	}

	/**
```

```
       * Callback operation used by the SpeechController when the user
       * has made an input.
       */
      public void resultPerformed(String recognizedText, Recognizer rec, int
objectId){
            if (this.hashCode() == objectId){
                  StringToSpeechInteractor confirmationInteractor = new
StringToSpeechInteractor(this.speechController, "You selected" +
recognizedText);
                  recognizedText = recognizedText.trim();
                  this.speechController.stopListening(this.speechRecognizer);
                  this.speechController.removeRule(ruleName);

                  if (recognizedText.equals("yes") ||
recognizedText.equals("check") || recognizedText.equals("checked")){
                        this.value = true;
                  } else if (recognizedText.equals("no") ||
recognizedText.equals("uncheck") || recognizedText.equals("not checked")){
                        this.value = false;
                  }
                  this.selectionPerformed(IActionListener.OK);
            }
      }
}
```

DateToSpeechInteractor.java

```
package jinteractorcontrols.speech;

import java.text.SimpleDateFormat;
import java.util.Locale;
import jinteractorcontrols.Date;
import jinteractorcontrols.DateToInteractor;

/**
 * Platform specific abstract class reflecting
 * the jinteractorcontrols.Date for the speech user
 * interface platform. It presents an
 * jinteractorcontrols.Date value for the user.
 */
public class DateToSpeechInteractor extends DateToInteractor{

      private StringToSpeechInteractor stringInteractor;
      private SpeechController speechController;
      private SimpleDateFormat formatter;
      private String formatPattern = "dd.MM.yyyy";

      /**
       * Constructor. Initializing the SpeechController.
       * @param speechController
       */
      public DateToSpeechInteractor(SpeechController speechController) {
            super();
            this.speechController = speechController;
            this.formatter = new SimpleDateFormat(formatPattern,
Locale.getDefault());
```

```
            stringInteractor = new
StringToSpeechInteractor(this.speechController);
      }

      /**
       * Constructor. Initializing the SpeechController and initial value.
       * @param speechController
       * @param value an initial Date value.
       */
      public DateToSpeechInteractor(SpeechController speechController, Date
value) {
            super(value);
            this.speechController = speechController;
            this.formatter = new SimpleDateFormat(formatPattern,
Locale.getDefault());
            stringInteractor = new
StringToSpeechInteractor(this.speechController,
formatter.format(this.value.getDate()));
      }

      /**
       * Re-reads the value.
       */
      public void refresh(){

      this.stringInteractor.setValue(formatter.format(this.value.getDate()));
      }
}
```

ToDateInteractor.java

```
package jinteractorcontrols.speech;

import java.text.SimpleDateFormat;
import java.util.Locale;
import javax.speech.recognition.Recognizer;
import jinteractorcontrols.IActionListener;

/**
 * Platform specific abstract class reflecting
 * the jinteractorcontrols.Date for the speech user
 * interface platform. It gets an
 * jinteractorcontrols.Date value as input
 * from the user.
 */
public class ToDateInteractor extends jinteractorcontrols.ToDateInteractor{

      private SpeechController speechController;
      private SpeechRecognizer speechRecognizer;

      private String monthRuleName = "month";
      private String dayRuleName = "day";
      private String yearRuleName = "year";
      private String confirmationText = "You selected ";
      private String invalidSelectionText = "Invalid selection.";
      private SimpleDateFormat formatter;
```

```
	/**
	 * Constructor. Initializing the SpeechController.
	 * @param speechController
	 */
	public ToDateInteractor(SpeechController speechController) {
		super();
		this.speechController = speechController;
		this.formatter = new SimpleDateFormat("dd.MM.yy",
Locale.getDefault());

	}

	/**
	 * Activates the interactor and makes it present
	 * it self for the user.
	 */
	public void activate(){
		startDayRecognizing();
	}

	/**
	 * Puts the interactor in listening mode
	 * and waits for the user to make a day selection.
	 */
	private void startDayRecognizing(){
		StringToSpeechInteractor descriptionInteractor = new
StringToSpeechInteractor(this.speechController, "Select a day: ");
		try{
			StringBuffer alternativesGrammar = new StringBuffer();
			for (int i = 1; i <= 31; i++){
				alternativesGrammar.append(i);
				if (i<31)
				alternativesGrammar.append("|");
			}
			this.speechController.addRule(dayRuleName,
alternativesGrammar.toString(), true);

			this.speechRecognizer = new
SpeechRecognizer(this.speechController.rec,
			new SpeechResultListener()
			{
				public void resultPerformed(String recognizedText,
Recognizer rec, int objectId){
					dayResultPerformed(recognizedText, rec,
objectId);
				}
			},
			this.hashCode());
			this.speechController.addResultListener(speechRecognizer);
			this.speechController.startListening();
		}
		catch (Exception e){
			e.printStackTrace();
		}
	}
```

```
	/**
	 * Callback operation used by the SpeechController when the user
	 * has made an day input.
	 */
	public void dayResultPerformed(String recognizedText, Recognizer rec,
int objectId){
		if (this.hashCode() == objectId){
			this.speechController.speak(this.confirmationText +
recognizedText);
			this.speechController.stopListening(this.speechRecognizer);
			this.speechController.removeRule(this.dayRuleName);

			recognizedText = recognizedText.trim();

	this.value.setDayOfMonth(Integer.decode(recognizedText).intValue());

			startMonthRecognizing();
		}
	}

	/**
	 * Puts the interactor in listening mode
	 * and waits for the user to make a month selection.
	 */
	private void startMonthRecognizing(){
		StringToSpeechInteractor descriptionInteractor = new
StringToSpeechInteractor(this.speechController, "Select a month: ");
		try{
			StringBuffer alternativesGrammar = new StringBuffer();
			for (int i = 1; i <= 12; i++){
				alternativesGrammar.append(i);
				if (i<12)
				alternativesGrammar.append("|");
			}
			this.speechController.addRule(monthRuleName,
alternativesGrammar.toString(), true);

			this.speechRecognizer = new
SpeechRecognizer(this.speechController.rec,
			new SpeechResultListener()
			{
				public void resultPerformed(String recognizedText,
Recognizer rec, int objectId){
					monthResultPerformed(recognizedText, rec,
objectId);
				}
			},
			this.hashCode());
			this.speechController.addResultListener(speechRecognizer);
			this.speechController.startListening();
		}
		catch (Exception e){
			e.printStackTrace();
		}
	}

	/**
```

```
       * Callback operation used by the SpeechController when the user
       * has made an month input.
       */
      public void monthResultPerformed(String recognizedText, Recognizer rec,
int objectId){
            if (this.hashCode() == objectId){
                  this.speechController.speak(this.confirmationText +
recognizedText);
                  this.speechController.stopListening(this.speechRecognizer);
                  this.speechController.removeRule(this.monthRuleName);

                  recognizedText = recognizedText.trim();

      this.value.setMonth(Integer.decode(recognizedText).intValue());

                  startYearRecognizing();
            }
      }

      /**
       * Puts the interactor in listening mode
       * and waits for the user to make a year selection.
       */
      private void startYearRecognizing(){
            StringToSpeechInteractor descriptionInteractor = new
StringToSpeechInteractor(this.speechController, "Select a year: ");
            try{
                  StringBuffer alternativesGrammar = new StringBuffer();
                  for (int i = 2000; i <= 2010; i++){
                        alternativesGrammar.append(i);
                        if (i<2010)
                        alternativesGrammar.append("|");
                  }
                  this.speechController.addRule(yearRuleName,
alternativesGrammar.toString(), true);

                  this.speechRecognizer = new
SpeechRecognizer(this.speechController.rec,
                  new SpeechResultListener()
                  {
                        public void resultPerformed(String recognizedText,
Recognizer rec, int objectId){
                              yearResultPerformed(recognizedText, rec,
objectId);
                        }
                  },
                  this.hashCode());
                  this.speechController.addResultListener(speechRecognizer);
                  this.speechController.startListening();
            }
            catch (Exception e){
                  e.printStackTrace();
            }
      }

      /**
       * Callback operation used by the SpeechController when the user
```

```
       * has made an year input.
       */
      public void yearResultPerformed(String recognizedText, Recognizer rec,
int objectId){
            if (this.hashCode() == objectId){
                  this.speechController.speak(this.confirmationText +
recognizedText);
                  this.speechController.stopListening(this.speechRecognizer);
                  this.speechController.removeRule(this.yearRuleName);

                  recognizedText = recognizedText.trim();

      this.value.setYear(Integer.decode(recognizedText).intValue());
            }

            this.speechController.speak("Selected date is: " +
formatter.format(this.value.getDate()));
            this.selectionPerformed(IActionListener.OK);
      }
}
```

IntegerToSpeechInteractor.java

```
package jinteractorcontrols.speech;

import jinteractorcontrols.IntegerToInteractor;

/**
 * Platform specific abstract class reflecting
 * the OCL::Integer for the speech user
 * interface platform. It presents an
 * OCL::Integer value for the user.
 */
public class IntegerToSpeechInteractor extends IntegerToInteractor{

      private StringToSpeechInteractor stringInteractor;
      private SpeechController speechController;

      /**
       * Constructor. Initializing the SpeechController.
       * @param speechController
       */
      public IntegerToSpeechInteractor(SpeechController speechController) {
            super();
            this.speechController = speechController;
            stringInteractor = new
StringToSpeechInteractor(this.speechController);
      }

      /**
       * Constructor. Initializing the SpeechController and initial value.
       * @param speechController
       * @param value an initial int value.
       */
      public IntegerToSpeechInteractor(SpeechController speechController, int
value) {
```

```
            super(value);
            this.speechController = speechController;
            stringInteractor = new
StringToSpeechInteractor(this.speechController,
Integer.toString(this.value));
      }

      /**
       * Re-reads the value.
       */
      public void refresh(){
            this.stringInteractor.setValue(Integer.toString(this.value));
      }
}
```

ToIntegerInteractor.java

```
package jinteractorcontrols.speech;

import javax.speech.recognition.Recognizer;
import javax.speech.synthesis.SpeakableListener;
import javax.speech.synthesis.Synthesizer;
import jinteractorcontrols.IActionListener;

/**
 * Platform specific abstract class reflecting
 * the OCL::Integer for the speech user
 * interface platform. It gets an
 * OCL::Integer value as input
 * from the user.
 */
public class ToIntegerInteractor extends
jinteractorcontrols.ToIntegerInteractor implements ISpeechResultListener {

      private Synthesizer synth;
      private SpeakableListener spList;
      private SpeechController speechController;
      private SpeechRecognizer speechRecognizer;

      private String ruleName = "integer";
      private String confirmationText = "You selected ";
      private String invalidSelectionText = "Invalid selection.";

      /**
       * Constructor. Initializing the SpeechController.
       * @param speechController
       */
      public ToIntegerInteractor(SpeechController speechController) {
            super();
            this.speechController = speechController;
      }

      /**
       * Activates the interactor and makes it present
       * it self for the user.
       */
```

```
	public void activate(){
		startSpeechRecognizing();
	}

	/**
	 * Puts the interactor in listening mode
	 * and waits for the user to make a selection.
	 */
	private void startSpeechRecognizing(){
		try{
			StringBuffer alternativesGrammar = new StringBuffer();
			for (int i = 0; i <= 1000; i++){
				alternativesGrammar.append(i + "|");
			}
			alternativesGrammar.append("1001");
			this.speechController.addRule("number",
alternativesGrammar.toString(), true);

			if (this.speechRecognizer == null)
				this.speechRecognizer = new
SpeechRecognizer(this.speechController.rec, this, this.hashCode());

	this.speechController.addResultListener(this.speechRecognizer);
			this.speechController.startListening();
		}
		catch (Exception e){
			e.printStackTrace();
		}
	}

	/**
	 * Callback operation used by the SpeechController when the user
	 * has made an input.
	 */
	public void resultPerformed(String recognizedText, Recognizer rec, int
objectId){
		if (this.hashCode() == objectId){
			this.speechController.speak(this.confirmationText +
recognizedText);
			this.speechController.stopListening(this.speechRecognizer);
			this.speechController.removeRule(this.ruleName);

			recognizedText = recognizedText.trim();
			this.value = Integer.decode(recognizedText).intValue();
			this.selectionPerformed(IActionListener.OK);
		}
	}
}
```

StringToSpeechInteractor.java

```
package jinteractorcontrols.speech;

import jinteractorcontrols.StringToInteractor;

/**
```

```
 * Platform specific abstract class reflecting
 * the OCL::String for the speech user
 * interface platform. It presents an
 * OCL::String value for the user.
 */
public class StringToSpeechInteractor extends StringToInteractor{

	private SpeechController speechController;

	/**
	 * Constructor. Initializing the SpeechController.
	 * @param speechController
	 */
	public StringToSpeechInteractor(SpeechController speechController){
		super();
		this.speechController = speechController;
	}

	/**
	 * Constructor. Initializing the SpeechController and initial value.
	 * @param speechController
	 * @param value an initial String value.
	 */
	public StringToSpeechInteractor(SpeechController speechController,
String value) {
		super(value);
		this.speechController = speechController;
		refresh();
	}

	/**
	 * Re-reads the value.
	 */
	public void refresh(){
		this.speechController.speak(this.value);
	}
}
```

ToStringInteractor.java

```
package jinteractorcontrols.speech;

import javax.speech.recognition.DictationGrammar;
import javax.speech.recognition.Recognizer;
import jinteractorcontrols.IActionListener;

/**
 * Platform specific abstract class reflecting
 * the OCL::String for the speech user
 * interface platform. It gets an
 * OCL::String value as input
 * from the user.
 */
public class ToStringInteractor extends
jinteractorcontrols.ToStringInteractor implements ISpeechResultListener {
```

```
	private SpeechController speechController;
	private SpeechRecognizer speechRecognizer;

	private String ruleName = "string";
	private String displayText = "Read in your value:";
	protected IActionListener listener;

	/**
	 * Constructor. Initializing the SpeechController.
	 * @param speechController
	 */
	public ToStringInteractor(SpeechController speechController) {
		super();
		this.speechController = speechController;
	}

	/**
	 * Activates the interactor and makes it present
	 * it self for the user.
	 */
	public void activate(){
		StringToSpeechInteractor displayTextInteractor = new
StringToSpeechInteractor(this.speechController, displayText);
		startSpeechRecognizing();
	}

	/**
	 * Puts the interactor in listening mode
	 * and waits for the user to make a selection.
	 */
	private void startSpeechRecognizing(){
		try{

			DictationGrammar dictation =
this.speechController.rec.getDictationGrammar("dictation");
			dictation.setEnabled(true);

			if (this.speechRecognizer == null)
				this.speechRecognizer = new
SpeechRecognizer(this.speechController.rec, this, this.hashCode());

	this.speechController.addResultListener(this.speechRecognizer);
			this.speechController.startListening();
		}
		catch (Exception e){
			e.printStackTrace();
		}
	}

	/**
	 * Callback operation used by the SpeechController when the user
	 * has made an input.
	 */
	public void resultPerformed(String recognizedText, Recognizer rec, int
objectId){
		if (this.hashCode() == objectId){
```

```
                    StringToSpeechInteractor confirmationInteractor = new
StringToSpeechInteractor(this.speechController, "Recognized value is " +
recognizedText);
                    recognizedText = recognizedText.trim();

                    DictationGrammar dictation =
this.speechController.rec.getDictationGrammar("dictation");
                    dictation.setEnabled(false);
                    this.speechController.stopListening(this.speechRecognizer);
                    this.value = recognizedText;
                    this.selectionPerformed(IActionListener.OK);
                }
            }
}
```

The Element Selection Pattern

Transformation rules (JavaElementSelection.xml)

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

public class <Element member="getName"/> extends
jinteractorcontrols.speech.ElementSelection {

        public <Element
member="getName"/>(jinteractorcontrols.speech.SpeechController
speechController){
          super(speechController);
        }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>

}
</Template>
```

ElementSelection.java

```
package jinteractorcontrols.speech;

import jinteractorcontrols.IActionListener;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Locale;
import javax.speech.recognition.Recognizer;
import javax.speech.synthesis.SpeakableListener;
import javax.speech.synthesis.Synthesizer;

/**
 * Platform specific abstract class reflecting
 * the ElementSelection pattern for the speech user
 * interface platform. It is a helper class inherited
 * by generated classes using the transformation rules
 * in JavaElementSelection.xml in the speech profile.
 */
```

```
public abstract class ElementSelection extends
jinteractorcontrols.ElementSelection implements ISpeechResultListener {

      private Synthesizer synth;
      private SpeakableListener spList;
      private SpeechController speechController;
      private SpeechRecognizer speechRecognizer;

      private String confirmationText = "You selected ";
      private String invalidSelectionText = "Invalid selection.";
      private String ruleName = "elementSelection";
      private String backCmdText = "Back";
      private String exitCmdText = "Exit";

      private ArrayList alternativeStrings;
      protected SimpleDateFormat formatter;

    /**
     * Constructor. Initializing the SpeechController.
     * @param speechController
     */
      public ElementSelection(SpeechController speechController) {
            super();
            this.speechController = speechController;

            this.formatter = new SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault());
      }

      /**
       * An operation that returns a string representing an element
       * in the set of available elements.
       * Must be implemented by all classes that
       * inherit this class.
       * @param element a element from the set.
       * @return string representing the element.
       */
      protected abstract String getDisplayName(Object element);

      /**
       * Callback operation used by the SpeechController when the user
       * has made an input.
       */
      public void resultPerformed(String recognizedText, Recognizer rec, int
objectId){
            if (this.hashCode() == objectId){
                  recognizedText = trim(recognizedText);

                  int index = -1;
                  try{
                        index = Integer.decode(recognizedText).intValue();
                  }catch (NumberFormatException e){
                  }


                  if (recognizedText.equals(this.backCmdText)){
```

```
      this.speechController.stopListening(this.speechRecognizer);
                        this.selectionPerformed(IActionListener.BACK);

                  } else if (recognizedText.equals(this.exitCmdText)){

      this.speechController.stopListening(this.speechRecognizer);
                        this.selectionPerformed(IActionListener.EXIT);
                  }else{
                        int responseIndex = -1;
                        if (index > 0){
                              responseIndex = index - 1;
                        }else{
                              for(int i = 0; i < alternativeStrings.size();
i++){
                                    String alternative =
((String)alternativeStrings.get(i));
                                    alternative = trim(alternative);
                                    if (recognizedText.equals(alternative)){
                                          responseIndex = i;
                                          break;
                                    }
                              }
                        }


      this.speechController.stopListening(this.speechRecognizer);

                        if ((responseIndex >= 0) && (responseIndex <
this.elementCollection.size())){
                              StringToSpeechInteractor
invalidSelectionInteractor = new
StringToSpeechInteractor(this.speechController, this.confirmationText +
Integer.toString(responseIndex+1) + ": " +
(String)this.alternativeStrings.get(responseIndex));

                              this.selectedElement =
this.elementCollection.get(responseIndex);
                              this.selectionPerformed(IActionListener.OK);
                        }
                        else
                        {
                              StringToSpeechInteractor
invalidSelectionInteractor = new
StringToSpeechInteractor(this.speechController, this.invalidSelectionText);
                              this.activate();
                        }
                  }
            }
      }

      /**
       * An trim method for rule strings
       * removing all spaces in the string.
       * @param s the rule string.
       * @return the trimed rule string.
       */
```

```
	private String trim(String s){
		String res = s.trim();
		res = res.replaceAll("\\s++", " ");
		return res;
	}

	/**
	 * Activates the interactor and makes it present
	 * it self for the user.
	 */
	public void activate(){
		speechController.speak(this.description);
		ArrayList displayStrings = new ArrayList();
		for (int i = 0; i < this.elementCollection.size(); i++){
			String displayText = "";
			Object element = this.elementCollection.get(i);

			displayText += getDisplayName(element);
			StringToSpeechInteractor displayTextInteractor = new
StringToSpeechInteractor(this.speechController, Integer.toString(i+1) + ": "
+ displayText);
			displayStrings.add(displayText);
		}

		//Add Back and Exit
		StringToSpeechInteractor displayTextInteractor = new
StringToSpeechInteractor(this.speechController, this.backCmdText);

		displayStrings.add(this.backCmdText);
		displayTextInteractor = new
StringToSpeechInteractor(this.speechController, this.exitCmdText);

		displayStrings.add(this.exitCmdText);

		this.alternativeStrings = displayStrings;
		startSpeechRecognizing();

	}

	/**
	 * Puts the interactor in listening mode
	 * and waits for the user to make a selection.
	 */
	private void startSpeechRecognizing(){
		try{
			String alternativesGrammar = "";
			for(int i = 0; i < this.alternativeStrings.size(); i++){
				alternativesGrammar +=
(String)this.alternativeStrings.get(i);

				alternativesGrammar += " | " + Integer.toString(i+1)
+ ": " + (String)this.alternativeStrings.get(i);
				alternativesGrammar += " | " + Integer.toString(i+1);

				if (i < this.alternativeStrings.size() - 1){
					alternativesGrammar += " | ";
				}
```

```
                  }

                  this.speechController.addRule(ruleName,
alternativesGrammar);
                  if (this.speechRecognizer == null)
                        this.speechRecognizer = new
SpeechRecognizer(this.speechController.rec, this, this.hashCode());

      this.speechController.addResultListener(this.speechRecognizer);
                  this.speechController.startListening();
            }
            catch (Exception e){
                  e.printStackTrace();
            }
      }
}
```

The Hierarchical Selection Pattern

Transformation rules (JavaHierarchicalSelection.xml)

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

import jinteractorcontrols.speech.SpeechController;

public class <Element member="getName"/> extends
jinteractorcontrols.speech.HierarchicalSelection {

        public class ESelection extends
jinteractorcontrols.speech.ElementSelection {

          public ESelection(SpeechController speechController){
            super(speechController);
          }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>
        }

        public <Element member="getName"/>(SpeechController
speechController){
            super(speechController);
                  initialize();
        }

              public void activate() {
                activateSelection((<Foreach
expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')"><Identifier
member="getType.getFullName" /></If></If></Foreach>)this.rootElement);
              }
```

```
        public void activateSelection(<Foreach
expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')"><Identifier
member="getType.getFullName" /></If></If></Foreach> element){
          ESelection selection = new ESelection(this.speechController);

          selection.setElements(element.getChildren());

          selection.setDescription(this.description);
          selection.setListener(new jinteractorcontrols.ActionListener() {
            public void actionPerformed(Object source, int type) {
              atcLevelSelectionPerformed(source, type);
            }
          });
           selection.activate();
        }

        public void atcLevelSelectionPerformed(Object source, int type){
          if (type == jinteractorcontrols.IActionListener.OK){
            jinteractorcontrols.ElementSelection selectionControl =
(jinteractorcontrols.ElementSelection) source;
            <Foreach expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')"><Identifier
member="getType.getFullName" /></If></If></Foreach> element
                  = (<Foreach expression="getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')"><Identifier
member="getType.getFullName" /></If></If></Foreach>)
selectionControl.getSelectedElement();
            if (element.isComposite()) {
              activateSelection(element);
            }
            else {
              this.selectedElement = element;

this.selectionPerformed(jinteractorcontrols.IActionListener.OK);
            }
          }
          else if (type == jinteractorcontrols.IActionListener.BACK) {

this.selectionPerformed(jinteractorcontrols.IActionListener.BACK);
          }
          else if (type == jinteractorcontrols.IActionListener.CANCEL) {

this.selectionPerformed(jinteractorcontrols.IActionListener.CANCEL);
          }
          else if (type == jinteractorcontrols.IActionListener.EXIT) {

this.selectionPerformed(jinteractorcontrols.IActionListener.EXIT);
          }
        }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>
```

```
}
</Template>
```

HierarchicalSelection.java

```
package jinteractorcontrols.speech;

/**
 * Platform specific abstract class reflecting
 * the HierarchicalSelection pattern for the speech user
 * interface platform. It is a helper class inherited
 * by generated classes using the transformation rules
 * in JavaHierarchicalSelection.xml in the speech profile.
 */
public abstract class HierarchicalSelection extends
jinteractorcontrols.HierarchicalSelection {

      protected SpeechController speechController;

      /**
       * Constructor. Initializing the SpeechController.
       * @param speechController
       */
      public HierarchicalSelection(SpeechController speechController) {
            super();
            this.speechController = speechController;
      }

      /**
       * Activates the interactor and makes it present
       * it self for the user. Must be implemented
       * by classes that inherit this class.
       */
      public abstract void activate();
}
```

The Interactor Selection Pattern

Transformation rules (JavaInteractorSelection.xml and JavaCommand.xml)

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

import jinteractorcontrols.speech.SpeechController;

public class <Element member="getName"/> extends
jinteractorcontrols.speech.InteractorSelection {


        public <Element member="getName"/>(SpeechController
speechController){
          super(speechController);

          initialize();
```

```
        }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>

}
</Template>
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

public class <Element member="getName"/> extends jinteractorcontrols.Command
{

        public <Element member="getName"/>(){
          initialize();
        }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>

}
</Template>
```

InteractorSelection.java

```
package jinteractorcontrols.speech;

import java.util.ArrayList;
import javax.speech.recognition.Recognizer;
import javax.speech.synthesis.SpeakableListener;
import javax.speech.synthesis.Synthesizer;
import jinteractorcontrols.Command;
import jinteractorcontrols.IActionListener;

/**
 * Platform specific abstract class reflecting
 * the InteractorSelection pattern for the speech user
 * interface platform. It is a helper class inherited
 * by generated classes using the transformation rules
 * in JavaInteractorSelection.xml in the speech profile.
 */
public class InteractorSelection extends
jinteractorcontrols.InteractorSelection implements ISpeechResultListener {

      private Synthesizer synth;
      private SpeakableListener spList;
      private SpeechController speechController;
      private SpeechRecognizer speechRecognizer;

      private String instructionText = "Select a command from the following
choices:";
      private String confirmationText = "You selected ";
      private String invalidSelectionText = "Invalid selection.";
```

```
	private String backCmdText = "Back";
	private String exitCmdText = "Exit";
	private String ruleName = "interacotrSelection";
	private Command backCmd;
	private Command exitCmd;

	/**
	 * Constructor. Initializing the SpeechController.
	 * @param speechController
	 */
	public InteractorSelection(SpeechController speechController) {
		super();
		this.speechController = speechController;

		this.backCmd = new Command(backCmdText);
		this.exitCmd = new Command(exitCmdText);

	}

	/**
	 * Callback operation used by the SpeechController when the user
	 * has made an input.
	 */
	public void resultPerformed(String recognizedText, Recognizer rec, int
objectId){
		if (this.hashCode() == objectId){

			int index = -1;
			try{
				Integer i = Integer.decode(recognizedText.trim());
				index = i.intValue();
			}catch (NumberFormatException e){
			}

			if (recognizedText.trim().equals(this.backCmdText)){

	this.speechController.stopListening(this.speechRecognizer);
				this.selectionPerformed(IActionListener.BACK);

			} else if (recognizedText.trim().equals(this.exitCmdText)){

	this.speechController.stopListening(this.speechRecognizer);
				this.selectionPerformed(IActionListener.EXIT);
			}else{
				int responseIndex = -1;

				if (index > 0){
					responseIndex = index - 1;
				}else{
					for(int i = 0; i <
this.commandCollection.size(); i++){
						String alternative =
((Command)this.commandCollection.get(i)).getName().trim();
						if
(recognizedText.trim().equals(alternative)){
							responseIndex = i;
							break;
```

```
                                    }
                              }
                        }


      this.speechController.stopListening(this.speechRecognizer);

                        if ((responseIndex >= 0) && (responseIndex <
this.commandCollection.size())){
                              this.speechController.speak(confirmationText +
Integer.toString(responseIndex+1) + ": "+
((Command)this.commandCollection.get(responseIndex)).getName());
                              this.selectedCommand =
(Command)this.commandCollection.get(responseIndex);
                              this.selectionPerformed(IActionListener.OK);
                        }
                        else
                        {

      this.speechController.speak(invalidSelectionText);
                              this.activate();
                        }
                  }
            }
      }

      /**
       * Activates the interactor and makes it present
       * it self for the user.
       */
      public void activate(){

            speechController.speak(instructionText);
            ArrayList displayStrings = new ArrayList();
            for (int i = 0; i < this.commandCollection.size(); i++){
                  Command cmd = (Command)this.commandCollection.get(i);

                  speechController.speak(Integer.toString(i+1) + ": "+
cmd.getName());
            }
            speechController.speak(this.backCmd.getName());
            speechController.speak(this.exitCmd.getName());

            startSpeechRecognizing();

      }

      /**
       * Puts the interactor in listening mode
       * and waits for the user to make a selection.
       */
      private void startSpeechRecognizing(){
            try{
                  String alternativesGrammar = "";
                  for(int i = 0; i < this.commandCollection.size(); i++){
                        alternativesGrammar +=
((Command)this.commandCollection.get(i)).getName();
```

```
                        alternativesGrammar += " | " + Integer.toString(i+1)
+ " " + ((Command)this.commandCollection.get(i)).getName();
                        alternativesGrammar += " | " + Integer.toString(i+1);

                        if (i < this.commandCollection.size() - 1)
                              alternativesGrammar += " | ";
                  }
                  alternativesGrammar += " | " + this.backCmd.getName();
                  alternativesGrammar += " | " + this.exitCmd.getName();

                  this.speechController.addRule(ruleName,
alternativesGrammar);
                  if (this.speechRecognizer == null)
                        this.speechRecognizer = new
SpeechRecognizer(this.speechController.rec, this, this.hashCode());

      this.speechController.addResultListener(this.speechRecognizer);
                  this.speechController.startListening();
            }
            catch (Exception e){
                  e.printStackTrace();
            }
      }

      /**
       * Set a instruction text to be used.
       * @param instruction the instruction.
       */
      public void setInstruction(String instruction){
            this.instructionText = instruction;
      }
}
```

The Edit/View Element Pattern

Transformation rules (JavaElementView.xml)

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<Template>
package <Element member="getNamespace.getFullName"/>;

import java.util.ArrayList;

import jinteractorcontrols.speech.SpeechController;
import jinteractorcontrols.speech.StringToSpeechInteractor;
import jinteractorcontrols.speech.ToBooleanInteractor;
import jinteractorcontrols.speech.ToStringInteractor;
import jinteractorcontrols.speech.ToDateInteractor;
import jinteractorcontrols.speech.DateToSpeechInteractor;

import jinteractorcontrols.ActionListener;
import jinteractorcontrols.IActionListener;
```

```
public class <Element member="getName"/> extends
jinteractorcontrols.EntityView {

            private SpeechController speechController;

        <Foreach expression="getOpositeAssociationEnds">
            private <Identifier member="getType.getFullName"/> element;
            </Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private boolean <Identifier member="getName"/>Visible = true;
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private boolean <Identifier member="getName"/>Enabled = true;
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private int <Identifier member="getName"/>Index = 0;
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private String <Identifier member="getName"/>Label = "<Identifier
member="getName"/>";
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private jinteractorcontrols.speech.<Identifier
member="getType.getName"/>ToSpeechInteractor <Identifier
member="getName"/>Interactor;
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private boolean <Identifier member="getName"/>Visible = true;
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private boolean <Identifier member="getName"/>Enabled = true;
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private int <Identifier member="getName"/>Index = 0;
            </If></If></Foreach></Foreach>
```

```
            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private String <Identifier member="getName"/>Label = "<Identifier
member="getName"/>";
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private jinteractorcontrols.speech.StringToSpeechInteractor
<Identifier member="getName"/>Interactor;
            </If></If></Foreach></Foreach>

            private ArrayList activatedControls;
            private int currentIndex;
            private int maxIndex = <Foreach
expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">1+</Foreach></Foreach><Foreach
expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">1+</If></If></Foreach
></Foreach>1;

        public <Element member="getName"/>(SpeechController speechController)
{
          this.speechController = speechController;
          initialize();
        }

        <Foreach expression="getOpositeAssociationEnds">
            public void set<Identifier member="getType.getName"/>(<Identifier
member="getType.getFullName"/> value) {this.element = value;}
            </Foreach>

        <Foreach expression="getOpositeAssociationEnds">
            public <Identifier member="getType.getFullName"/> get<Identifier
member="getType.getName"/>() { return this.element;}
            </Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>V
isible(boolean value) { this.<Identifier member="getName"/>Visible = value; }
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>E
nabled(boolean value) { this.<Identifier member="getName"/>Enabled = value; }
            </Foreach></Foreach>
```

```
            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>I
ndex(int value) { this.<Identifier member="getName"/>Index = value; }
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>L
abel(String value) { this.<Identifier member="getName"/>Label = value; }
            </Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>V
isible(boolean value) { this.<Identifier member="getName"/>Visible = value; }
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>E
nabled(boolean value) { this.<Identifier member="getName"/>Enabled = value; }
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            public void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>I
ndex(int value) { this.<Identifier member="getName"/>Index = value; }
            </If></If></Foreach></Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private void set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>L
abel(String value) { this.<Identifier member="getName"/>Label = value; }
            </If></If></Foreach></Foreach>


              public void activate(){
                  this.activatedControls = new ArrayList();
                  currentIndex = -1;
                  activateNext();
              }
```

```
              public void activateNext(){
                  if (currentIndex &lt; maxIndex){
                        currentIndex++;

                        <Foreach
expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
                        if (this.<Identifier member="getName"/>Visible
&amp;&amp; this.<Identifier member="getName"/>Index == this.currentIndex
&amp;&amp; !this.activatedControls.contains("<Identifier
member="getName"/>"))
                        {
                              activate<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
);
                              this.activatedControls.add("<Identifier
member="getName"/>");
                              return;
                        }
                        </Foreach></Foreach>
                        <Foreach
expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
                        if (this.<Identifier member="getName"/>Visible
&amp;&amp; this.<Identifier member="getName"/>Index == this.currentIndex
&amp;&amp; !this.activatedControls.contains("<Identifier
member="getName"/>"))
                        {
                              activate<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
);
                              this.activatedControls.add("<Identifier
member="getName"/>");
                              return;
                        }
                        </If></If></Foreach></Foreach>
                        activateNext();
                  }
                  else{
                        finished();
                  }

              }

        <Foreach expression="getConstraints">
            <Identifier member="getJavaExpression" />
            </Foreach>

            <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getAttributes">
            private void activate<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
){
            if (<Identifier member="getName" />Visible){
```

```
                String description = "Attribute " + <Identifier
member="getName"/>Label + " contains the value: ";
                String emptyDescription = "Attribute " + <Identifier
member="getName" />Label + " dosage contains no value. ";
                String changeDescription = "Do you want to change this
value? ";

                StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                <Identifier member="getType.getName" />ToSpeechInteractor
valueStringTo = new <Identifier member="getType.getName"
/>ToSpeechInteractor(this.speechController);

                <If expression="getType.getName.endsWith('String')">
                if (this.element.get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
) == null || this.element.get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
) == ""){
                </If>
                <If expression="getType.getName.endsWith('Date')">
                if (this.element.get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
) == null){
                </If>
                      descriptionStringTo.setValue(emptyDescription);
                }else{
                      descriptionStringTo.setValue(description);
                      valueStringTo.setValue(this.element.get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
));
                }

                if (<Identifier member="getName" />Enabled){
                      StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                      actionStringTo.setValue(changeDescription);

                      ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                      doChange.setListener(new ActionListener() {
                      public void actionPerformed(Object source, int type)
{
                            if (type == IActionListener.OK){
                              ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                              boolean value = doChange.getValue();
                              if (value){
                                  get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
);
                              } else {
                                  activateNext();
                              }
                            }
                      }
                      });
```

```
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
                  activateNext();
            }
      }

      private void get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
){
            To<Identifier member="getType.getName" />Interactor interactor =
new To<Identifier member="getType.getName"
/>Interactor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              <Identifier member="getName" />Changed(source, type);
            }
            });
            interactor.activate();

      }

      public void <Identifier member="getName" />Changed(Object source, int
type){
        if (type == IActionListener.OK){
            To<Identifier member="getType.getName" />Interactor interactor =
(To<Identifier member="getType.getName" />Interactor)source;
            this.element.set<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
interactor.getValue());
            activateNext();
        }
      }
      </Foreach></Foreach>

      <Foreach expression="getOpositeAssociationEnds"><Foreach
expression="getType.getOpositeAssociationEnds"><If
expression="getMultiplicity.startsWith('1')"><If
expression="getVisibility.getName.startsWith('public')">
            private void activate<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>(
){
            if (<Identifier member="getName" />Visible){
                  String description = "Attribute " + <Identifier
member="getName"/>Label + " contains the value: ";
                  String emptyDescription = "Attribute " + <Identifier
member="getName" />Label + " dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);
```

```
                  if (get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>D
isplayName(this.element) == null || get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>D
isplayName(this.element) == ""){
                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);
                        valueStringTo.setValue(get<Identifier
member="getName.substring(0,1).toUpperCase().concat(getName.substring(1))"/>D
isplayName(this.element));
                  }

            }else{
                  activateNext();
            }
      }
    </If></If></Foreach></Foreach>


      private void finished(){
            this.selectionPerformed(IActionListener.OK);
      }
}
</Template>
```

EntityView.java

```
package jinteractorcontrols.speech;

import java.util.Vector;

/**
 * Platform specific abstract class reflecting
 * the Edit/View Element pattern for the speech user
 * interface platform. It is a helper class inherited
 * by generated classes using the transformation rules
 * in JavaElementView.xml in the speech profile.
 */
public abstract class EntityView extends jinteractorcontrols.EntityView {

      /**
       * Constructor.
       */
      public EntityView() {
            super();
      }
}
```

Case study source files

Mobile Phone Client

The following section includes the generated source code for the mobile phone user interface interactors.

AppointmentSelection.java

package medication.ui.midp;

```
import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class AppointmentSelection extends jinteractorcontrols.midp.MIDPElementSelection {

       public AppointmentSelection(Form parentForm, MIDlet midlet){
         super(parentForm, midlet);
         initialize();
       }

       public String getDisplayName(Object element){
         return ((medication.businessEntities.Appointment)element).getStartTime().toTimeString().concat("
").concat(((medication.businessEntities.Appointment)element).getPatient().getLastName()).concat("
").concat(((medication.businessEntities.Appointment)element).getPatient().getFirstName());
             }

        public void initialize(){
         description = "Select Appointment:";
       }
}
```

AtcLevelSelection.java

```
package medication.ui.midp;

import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class AtcLevelSelection extends jinteractorcontrols.midp.MIDPHierarchicalSelection {

             public class ESelection extends jinteractorcontrols.midp.MIDPElementSelection {

               public ESelection(Form parentForm, MIDlet midlet){
                 super(parentForm, midlet);
               }

            public String getDisplayName(Object element){
          return ((medication.businessEntities.AtcLevel)element).getCode().concat("
").concat(((medication.businessEntities.AtcLevel)element).getName());
              }

     public void initialize(){
        description = "Select ATC:";
      }
           }

           private Form parentForm;
           private MIDlet midlet;

      public AtcLevelSelection(Form parentForm, MIDlet midlet){
                      this.parentForm = parentForm;
                      this.midlet = midlet;
                  initialize();
```

```
    }

    public void activate() {
          activateSelection((medication.businessEntities.AtcLevel)this.rootElement);
    }

    public void activateSelection(medication.businessEntities.AtcLevel element){
      Form form = new Form(this.description);
      ESelection selection = new ESelection(form,
          this.midlet);

      selection.setElements(element.getChildren());

      selection.setDescription(this.description);
      selection.setListener(new jinteractorcontrols.ActionListener() {
        public void actionPerformed(Object source, int type) {
          atcLevelSelectionPerformed(source, type);
        }
      });
      javax.microedition.lcdui.Display.getDisplay(this.midlet).setCurrent(form);
      selection.activate();
    }

    public void atcLevelSelectionPerformed(Object source, int type){
      if (type == jinteractorcontrols.IActionListener.OK){
        jinteractorcontrols.ElementSelection selectionControl = (jinteractorcontrols.ElementSelection)
source;
          medication.businessEntities.AtcLevel element
           = (medication.businessEntities.AtcLevel) selectionControl.getSelectedElement();
          if (element.isComposite()) {
            activateSelection(element);
          }
          else {
            this.selectedElement = element;
            this.selectionPerformed(jinteractorcontrols.IActionListener.OK);
          }
        }
        else if (type == jinteractorcontrols.IActionListener.BACK) {
          this.selectionPerformed(jinteractorcontrols.IActionListener.BACK);
        }
        else if (type == jinteractorcontrols.IActionListener.CANCEL) {
          this.selectionPerformed(jinteractorcontrols.IActionListener.CANCEL);
        }
        else if (type == jinteractorcontrols.IActionListener.EXIT) {
          this.selectionPerformed(jinteractorcontrols.IActionListener.EXIT);
        }
    }

          public String getDisplayName(Object element){
        return ((medication.businessEntities.AtcLevel)element).getCode().concat("
").concat(((medication.businessEntities.AtcLevel)element).getName());
            }
```

```
        public void initialize(){
          description = "Select ATC:";
        }
}
```

DateSelection.java

```
package medication.ui.midp;

import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class DateSelection extends jinteractorcontrols.midp.MIDPElementSelection {

        public DateSelection(Form parentForm, MIDlet midlet){
          super(parentForm, midlet);
          initialize();
        }


      public String getDisplayName(Object element){
          return ((jinteractorcontrols.Date)element).toDateString();
        }

       public void initialize(){
          description = "Select a Date:";
        }
}
```

MedicationView.java

```
package medication.ui.midp;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class MedicationView extends jinteractorcontrols.EntityView implements CommandListener {

                        private MIDlet midlet;
                        private Form form;
                        private jinteractorcontrols.IActionListener listener;
                        private medication.businessEntities.Medication element;
                        private boolean dosageVisible = true;
                        private boolean startDateVisible = true;
                        private boolean endDateVisible = true;
                        private boolean instructionVisible = true;
                        private boolean ordinatorVisible = true;
                        private boolean seponatorVisible = true;
                        private boolean dosageEnabled = true;
                        private boolean startDateEnabled = true;
                        private boolean endDateEnabled = true;
                        private boolean instructionEnabled = true;
                        private boolean ordinatorEnabled = true;
```

```
                private boolean seponatorEnabled = true;
                private int dosageIndex = 0;
                private int startDateIndex = 0;
                private int endDateIndex = 0;
                private int instructionIndex = 0;
                private int ordinatorIndex = 0;
                private int seponatorIndex = 0;
                private String dosageLabel = "dosage";
                private String startDateLabel = "startDate";
                private String endDateLabel = "endDate";
                private String instructionLabel = "instruction";
                private String ordinatorLabel = "ordinator";
                private String seponatorLabel = "seponator";
                private jinteractorcontrols.midp.MIDPStringToInteractor dosageInteractorR;
                private jinteractorcontrols.midp.MIDPToStringInteractor dosageInteractorW;
                private jinteractorcontrols.midp.MIDPDateToInteractor startDateInteractorR;
                private jinteractorcontrols.midp.MIDPToDateInteractor startDateInteractorW;
                private jinteractorcontrols.midp.MIDPDateToInteractor endDateInteractorR;
                private jinteractorcontrols.midp.MIDPToDateInteractor endDateInteractorW;
                private jinteractorcontrols.midp.MIDPStringToInteractor instructionInteractorR;
                private jinteractorcontrols.midp.MIDPToStringInteractor instructionInteractorW;
                private jinteractorcontrols.midp.MIDPStringToInteractor ordinatorInteractorR;
                private jinteractorcontrols.midp.MIDPToStringInteractor ordinatorInteractorW;
                private jinteractorcontrols.midp.MIDPStringToInteractor seponatorInteractorR;
                private jinteractorcontrols.midp.MIDPToStringInteractor seponatorInteractorW;
                private boolean drugVisible = true;
                private boolean drugEnabled = true;
                private int drugIndex = 0;
                private String drugLabel = "drug";
                private jinteractorcontrols.midp.MIDPStringInteractor drugInteractor;

        public MedicationView(MIDlet midlet) {
          this.midlet = midlet;
          initialize();
          this.form = new Form(this.description);

          this.form.addCommand(new javax.microedition.lcdui.Command("Back",
javax.microedition.lcdui.Command.BACK, 0));
                this.form.addCommand(new javax.microedition.lcdui.Command("Ok",
javax.microedition.lcdui.Command.OK, 0));
                this.form.addCommand(new javax.microedition.lcdui.Command("Cancel",
javax.microedition.lcdui.Command.CANCEL, 1));
                this.form.setCommandListener(this);
        }

                public void setMedication(medication.businessEntities.Medication value) {this.element =
value;}
                public medication.businessEntities.Medication getMedication() { updateElement(); return
this.element;}
                public void setDosageVisible(boolean value) { this.dosageVisible = value; }
                public void setStartDateVisible(boolean value) { this.startDateVisible = value; }
                public void setEndDateVisible(boolean value) { this.endDateVisible = value; }
                public void setInstructionVisible(boolean value) { this.instructionVisible = value; }
                public void setOrdinatorVisible(boolean value) { this.ordinatorVisible = value; }
```

```
                public void setSeponatorVisible(boolean value) { this.seponatorVisible = value; }
                public void setDosageEnabled(boolean value) { this.dosageEnabled = value; }
                public void setStartDateEnabled(boolean value) { this.startDateEnabled = value; }
                public void setEndDateEnabled(boolean value) { this.endDateEnabled = value; }
                public void setInstructionEnabled(boolean value) { this.instructionEnabled = value; }
                public void setOrdinatorEnabled(boolean value) { this.ordinatorEnabled = value; }
                public void setSeponatorEnabled(boolean value) { this.seponatorEnabled = value; }
                public void setDosageIndex(int value) { this.dosageIndex = value; }
                public void setStartDateIndex(int value) { this.startDateIndex = value; }
                public void setEndDateIndex(int value) { this.endDateIndex = value; }
                public void setInstructionIndex(int value) { this.instructionIndex = value; }
                public void setOrdinatorIndex(int value) { this.ordinatorIndex = value; }
                public void setSeponatorIndex(int value) { this.seponatorIndex = value; }
                private void setDosageLabel(String value) { this.dosageLabel = value; }
                private void setStartDateLabel(String value) { this.startDateLabel = value; }
                private void setEndDateLabel(String value) { this.endDateLabel = value; }
                private void setInstructionLabel(String value) { this.instructionLabel = value; }
                private void setOrdinatorLabel(String value) { this.ordinatorLabel = value; }
                private void setSeponatorLabel(String value) { this.seponatorLabel = value; }
                public void setDrugVisible(boolean value) { this.drugVisible = value; }
                public void setDrugEnabled(boolean value) { this.drugEnabled = value; }
                public void setDrugIndex(int value) { this.drugIndex = value; }
                private void setDrugLabel(String value) { this.drugLabel = value; }


                private void createInteractors(int index){

        if (dosageVisible && dosageIndex == index){
          if (dosageEnabled)
                        dosageInteractorW = new
jinteractorcontrols.midp.MIDPToStringInteractor(this.form, this.dosageLabel, this.element.getDosage()) ;
                        else
                        dosageInteractorR = new
jinteractorcontrols.midp.MIDPStringToInteractor(this.form, this.dosageLabel, this.element.getDosage()) ;

                }

        if (startDateVisible && startDateIndex == index){
          if (startDateEnabled)
                        startDateInteractorW = new
jinteractorcontrols.midp.MIDPToDateInteractor(this.form, this.startDateLabel, this.element.getStartDate())
;
                        else
                        startDateInteractorR = new
jinteractorcontrols.midp.MIDPDateToInteractor(this.form, this.startDateLabel, this.element.getStartDate())
;
                }

        if (endDateVisible && endDateIndex == index){
          if (endDateEnabled)
                        endDateInteractorW = new
jinteractorcontrols.midp.MIDPToDateInteractor(this.form, this.endDateLabel, this.element.getEndDate()) ;
                        else
                        endDateInteractorR = new
```

```
jinteractorcontrols.midp.MIDPDateToInteractor(this.form, this.endDateLabel, this.element.getEndDate()) ;

                    }

          if (instructionVisible && instructionIndex == index){
            if (instructionEnabled)
                        instructionInteractorW = new
jinteractorcontrols.midp.MIDPToStringInteractor(this.form, this.instructionLabel,
this.element.getInstruction()) ;
                        else
                        instructionInteractorR = new
jinteractorcontrols.midp.MIDPStringToInteractor(this.form, this.instructionLabel,
this.element.getInstruction()) ;
                    }

          if (ordinatorVisible && ordinatorIndex == index){
            if (ordinatorEnabled)
                        ordinatorInteractorW = new
jinteractorcontrols.midp.MIDPToStringInteractor(this.form, this.ordinatorLabel,
this.element.getOrdinator()) ;
                        else
                        ordinatorInteractorR = new
jinteractorcontrols.midp.MIDPStringToInteractor(this.form, this.ordinatorLabel,
this.element.getOrdinator()) ;
                    }

          if (seponatorVisible && seponatorIndex == index){
            if (seponatorEnabled)
                        seponatorInteractorW = new
jinteractorcontrols.midp.MIDPToStringInteractor(this.form, this.seponatorLabel,
this.element.getSeponator()) ;
                        else
                        seponatorInteractorR = new
jinteractorcontrols.midp.MIDPStringToInteractor(this.form, this.seponatorLabel,
this.element.getSeponator()) ;
                    }

          if (drugVisible && drugIndex == index)
                    drugInteractor = new jinteractorcontrols.midp.MIDPStringInteractor(this.form,
this.drugLabel, getDrugDisplayName((Object)this.element.getDrug()), 150, false) ;
                   }

                  private void updateElement(){

          if (dosageVisible && dosageEnabled)
                    element.setDosage(dosageInteractorW.getValue());

          if (startDateVisible && startDateEnabled)
                    element.setStartDate(startDateInteractorW.getValue());

          if (endDateVisible && endDateEnabled)
                    element.setEndDate(endDateInteractorW.getValue());

          if (instructionVisible && instructionEnabled)
```

```
                    element.setInstruction(instructionInteractorW.getValue());

            if (ordinatorVisible && ordinatorEnabled)
                    element.setOrdinator(ordinatorInteractorW.getValue());

            if (seponatorVisible && seponatorEnabled)
                    element.setSeponator(seponatorInteractorW.getValue());

                }

                  public void activate(){
                          for (int i = 1; i < 1+1+1+1+1+1+1+1+1; i++)
                                  createInteractors(i-1);
                    Display.getDisplay(this.midlet).setCurrent(this.form);
                  }

                  public void setListener(jinteractorcontrols.IActionListener listener){
                    this.listener = listener;
                  }

                  public void commandAction(javax.microedition.lcdui.Command lcduiCmd, Displayable
displayable){
                    if (lcduiCmd.getCommandType() == javax.microedition.lcdui.Command.BACK){
                      listener.actionPerformed(this, jinteractorcontrols.IActionListener.BACK);
                    }
                    if (lcduiCmd.getCommandType() == javax.microedition.lcdui.Command.CANCEL){
                      listener.actionPerformed(this, jinteractorcontrols.IActionListener.CANCEL);
                    }
                    else
                      listener.actionPerformed(this, jinteractorcontrols.IActionListener.OK);
                  }

                public void initialize(){
        description = "Medication";
        drugIndex = 0;
        startDateIndex = 1;
        endDateIndex = 2;
        dosageIndex = 3;
        instructionIndex = 4;
        ordinatorVisible = false;
        seponatorVisible = false;
        dosageEnabled = false;
        startDateEnabled = false;
        instructionEnabled = false;
        drugLabel = "Drug:";
        dosageLabel = "Dosage:";
        startDateLabel = "OrdinationDate:";
        endDateLabel = "SeponationDate:";
        instructionLabel = "Instruction:";
      }

                public String getDrugDisplayName(Object element){
        return ((medication.businessEntities.Drug)element).getCode().concat("
").concat(((medication.businessEntities.Drug)element).getName());
```

```
            }
}
```

MedicationSelection.java

```
package medication.ui.midp;

import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class MedicationSelection extends jinteractorcontrols.midp.MIDPElementSelection {

      public MedicationSelection(Form parentForm, MIDlet midlet){
        super(parentForm, midlet);

        initialize();

      }

                  public String getDisplayName(Object element){
        return ((medication.businessEntities.Medication)element).getStartDate().toString().concat("
").concat(((medication.businessEntities.Medication)element).getDrug().getName());
            }

                  public void initialize(){
        description = "Medications";
      }
}
```

PatientView.java

```
package medication.ui.midp;

import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class PatientView extends jinteractorcontrols.midp.MIDPInteractorSelection {


      public PatientView(Form parentForm){
        super(parentForm);

        initialize();

      }
                  public void initialize(){ ; }
}
```

MedicationSelectionViewCommand.java

```
package medication.ui.midp;

public class MedicationsSelectionViewCommand extends jinteractorcontrols.Command {
```

```
	public MedicationsSelectionViewCommand(){
	  initialize();
	}
		public void initialize(){
	  name = "View";
	}
}
```

OrdinateCommand.java

```
package medication.ui.midp;

public class OrdinateCommand extends jinteractorcontrols.Command {

	public OrdinateCommand(){
	  initialize();
	}
		public void initialize(){
	  name = "Ordinate";
	}

}
```

PatientRetrieval.java

```
package medication.ui.midp;

import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class PatientRetrieval extends jinteractorcontrols.midp.MIDPInteractorSelection {

	public PatientRetrieval(Form parentForm){
	  super(parentForm);

	  initialize();

	}
		public void initialize(){; }
}
```

PatientSearchCommand.java

```
package medication.ui.midp;

public class PatientSearchCommand extends jinteractorcontrols.Command {

	public PatientSearchCommand(){
	  initialize();
	}
		public void initialize(){
	  name = "Patient Search";
	}
```

```
}
```

PatientRetrievalByAppointmentCommand.java

```
package medication.ui.midp;

public class PatientRetrievalByAppointmentCommand extends jinteractorcontrols.Command {

      public PatientRetrievalByAppointmentCommand(){
        initialize();
      }
                  public void initialize(){
        name = "Appointments";
      }
}
```

PatientSearchCriteriaVew.java

```
package medication.ui.midp;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class PatientSearchCriteriaView extends jinteractorcontrols.EntityView implements
CommandListener {

                  private MIDlet midlet;
                  private Form form;
                  private jinteractorcontrols.IActionListener listener;
                  private medication.businessEntities.PatientSearchCriteria element;
                  private boolean searchStringVisible = true;
                  private boolean searchStringEnabled = true;
                  private int searchStringIndex = 0;
                  private String searchStringLabel = "searchString";
                  private jinteractorcontrols.midp.MIDPStringToInteractor searchStringInteractorR;
                  private jinteractorcontrols.midp.MIDPToStringInteractor searchStringInteractorW;
                  private boolean patientSearchTypeVisible = true;
                  private boolean patientSearchTypeEnabled = true;
                  private int patientSearchTypeIndex = 0;
                  private String patientSearchTypeLabel = "patientSearchType";
                  private jinteractorcontrols.midp.MIDPStringInteractor patientSearchTypeInteractor;

      public PatientSearchCriteriaView(MIDlet midlet) {
        this.midlet = midlet;
        initialize();
        this.form = new Form(this.description);

        this.form.addCommand(new javax.microedition.lcdui.Command("Back",
javax.microedition.lcdui.Command.BACK, 0));
            this.form.addCommand(new javax.microedition.lcdui.Command("Ok",
javax.microedition.lcdui.Command.OK, 0));
            this.form.addCommand(new javax.microedition.lcdui.Command("Cancel",
javax.microedition.lcdui.Command.CANCEL, 1));
```

```
            this.form.setCommandListener(this);
      }


                  public void setPatientSearchCriteria(medication.businessEntities.PatientSearchCriteria
value) {this.element = value;}
                  public medication.businessEntities.PatientSearchCriteria getPatientSearchCriteria() {
updateElement(); return this.element;}
                  public void setSearchStringVisible(boolean value) { this.searchStringVisible = value; }
                  public void setSearchStringEnabled(boolean value) { this.searchStringEnabled = value; }
                  public void setSearchStringIndex(int value) { this.searchStringIndex = value; }
                  private void setSearchStringLabel(String value) { this.searchStringLabel = value; }
                  public void setPatientSearchTypeVisible(boolean value) { this.patientSearchTypeVisible
= value; }
                  public void setPatientSearchTypeEnabled(boolean value) {
this.patientSearchTypeEnabled = value; }
                  public void setPatientSearchTypeIndex(int value) { this.patientSearchTypeIndex = value;
}
                  private void setPatientSearchTypeLabel(String value) { this.patientSearchTypeLabel =
value; }
                  private void createInteractors(int index){

            if (searchStringVisible && searchStringIndex == index){
              if (searchStringEnabled)
                        searchStringInteractorW = new
jinteractorcontrols.midp.MIDPToStringInteractor(this.form, this.searchStringLabel,
this.element.getSearchString()) ;
                        else
                        searchStringInteractorR = new
jinteractorcontrols.midp.MIDPStringToInteractor(this.form, this.searchStringLabel,
this.element.getSearchString()) ;
                     }



            if (patientSearchTypeVisible && patientSearchTypeIndex == index)
                  patientSearchTypeInteractor = new
jinteractorcontrols.midp.MIDPStringInteractor(this.form, this.patientSearchTypeLabel,
getPatientSearchTypeDisplayName((Object)this.element.getPatientSearchType()), 150, false) ;


                  }

                  private void updateElement(){

            if (searchStringVisible && searchStringEnabled)
                  element.setSearchString(searchStringInteractorW.getValue());

                  }

                    public void activate(){
                        for (int i = 1; i < 1+1+1; i++)
                              createInteractors(i-1);
                     Display.getDisplay(this.midlet).setCurrent(this.form);
```

```
                    }

                    public void setListener(jinteractorcontrols.IActionListener listener){
                      this.listener = listener;
                    }

                    public void commandAction(javax.microedition.lcdui.Command lcduiCmd, Displayable
displayable){
                      if (lcduiCmd.getCommandType() == javax.microedition.lcdui.Command.BACK){
                        listener.actionPerformed(this, jinteractorcontrols.IActionListener.BACK);
                      }
                      if (lcduiCmd.getCommandType() == javax.microedition.lcdui.Command.CANCEL){
                        listener.actionPerformed(this, jinteractorcontrols.IActionListener.CANCEL);
                      }
                      else
                        listener.actionPerformed(this, jinteractorcontrols.IActionListener.OK);
                    }

                  public void initialize(){
      description = "SetQueryString:";
      patientSearchTypeVisible = false;
      searchStringEnabled = true;
      searchStringVisible = true;
      searchStringIndex = 0;
      searchStringLabel = "Query:";
    }
                  public String getPatientSearchTypeDisplayName(Object element){
      return ((medication.businessEntities.PatientSearchType)element).getName();
    }
}
```

PatientSearchTypeSelection.java

```
package medication.ui.midp;

import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class PatientSearchTypeSelection extends jinteractorcontrols.midp.MIDPElementSelection {

    public PatientSearchTypeSelection(Form parentForm, MIDlet midlet){
      super(parentForm, midlet);
      initialize();
    }

                  public String getDisplayName(Object element){
      return ((medication.businessEntities.PatientSearchType)element).getName();
    }

                  public void initialize(){
      description = "Select a Search Type:";
    }
}
```

PatientSelection.java

```
package medication.ui.midp;

import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;

public class PatientSelection extends jinteractorcontrols.midp.MIDPElementSelection {

	public PatientSelection(Form parentForm, MIDlet midlet){
	  super(parentForm, midlet);

	  initialize();

	}
			public String getDisplayName(Object element){
	  return ((medication.businessEntities.Patient)element).getLastName().concat(",
").concat(((medication.businessEntities.Patient)element).getFirstName()).concat("
").concat(((medication.businessEntities.Patient)element).getMiddleName()).concat("
").concat(((medication.businessEntities.Patient)element).getDateOfBirth().toDateString());
		}
			public void initialize(){
	  description = "Select Patient:";
	}
}
```

MedicationCreation.java

```
package medication.ui.midp;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class MedicationCreation extends jinteractorcontrols.EntityView implements CommandListener {

			private MIDlet midlet;
			private Form form;
			private jinteractorcontrols.IActionListener listener;
			private medication.businessEntities.Medication element;
			private boolean dosageVisible = true;
			private boolean startDateVisible = true;
			private boolean endDateVisible = true;
			private boolean instructionVisible = true;
			private boolean ordinatorVisible = true;
			private boolean seponatorVisible = true;
			private boolean dosageEnabled = true;
			private boolean startDateEnabled = true;
			private boolean endDateEnabled = true;
			private boolean instructionEnabled = true;
			private boolean ordinatorEnabled = true;
			private boolean seponatorEnabled = true;
			private int dosageIndex = 0;
```

```
                private int startDateIndex = 0;
                private int endDateIndex = 0;
                private int instructionIndex = 0;
                private int ordinatorIndex = 0;
                private int seponatorIndex = 0;
                private String dosageLabel = "dosage";
                private String startDateLabel = "startDate";
                private String endDateLabel = "endDate";
                private String instructionLabel = "instruction";
                private String ordinatorLabel = "ordinator";
                private String seponatorLabel = "seponator";
                private jinteractorcontrols.midp.MIDPStringToInteractor dosageInteractorR;
                private jinteractorcontrols.midp.MIDPToStringInteractor dosageInteractorW;
                private jinteractorcontrols.midp.MIDPDateToInteractor startDateInteractorR;
                private jinteractorcontrols.midp.MIDPToDateInteractor startDateInteractorW;
                private jinteractorcontrols.midp.MIDPDateToInteractor endDateInteractorR;
                private jinteractorcontrols.midp.MIDPToDateInteractor endDateInteractorW;
                private jinteractorcontrols.midp.MIDPStringToInteractor instructionInteractorR;
                private jinteractorcontrols.midp.MIDPToStringInteractor instructionInteractorW;
                private jinteractorcontrols.midp.MIDPStringToInteractor ordinatorInteractorR;
                private jinteractorcontrols.midp.MIDPToStringInteractor ordinatorInteractorW;
                private jinteractorcontrols.midp.MIDPStringToInteractor seponatorInteractorR;
                private jinteractorcontrols.midp.MIDPToStringInteractor seponatorInteractorW;
                private boolean drugVisible = true;
                private boolean drugEnabled = true;
                private int drugIndex = 0;
                private String drugLabel = "drug";
                private jinteractorcontrols.midp.MIDPStringInteractor drugInteractor;

        public MedicationCreation(MIDlet midlet) {
          this.midlet = midlet;
          initialize();
          this.form = new Form(this.description);

          this.form.addCommand(new javax.microedition.lcdui.Command("Back",
javax.microedition.lcdui.Command.BACK, 0));
                this.form.addCommand(new javax.microedition.lcdui.Command("Ok",
javax.microedition.lcdui.Command.OK, 0));
                this.form.addCommand(new javax.microedition.lcdui.Command("Cancel",
javax.microedition.lcdui.Command.CANCEL, 1));
                this.form.setCommandListener(this);
        }

                public void setMedication(medication.businessEntities.Medication value) {this.element =
value;}
                public medication.businessEntities.Medication getMedication() { updateElement(); return
this.element;}
                public void setDosageVisible(boolean value) { this.dosageVisible = value; }
                public void setStartDateVisible(boolean value) { this.startDateVisible = value; }
                public void setEndDateVisible(boolean value) { this.endDateVisible = value; }
                public void setInstructionVisible(boolean value) { this.instructionVisible = value; }
                public void setOrdinatorVisible(boolean value) { this.ordinatorVisible = value; }
                public void setSeponatorVisible(boolean value) { this.seponatorVisible = value; }
                public void setDosageEnabled(boolean value) { this.dosageEnabled = value; }
```

```
                public void setStartDateEnabled(boolean value) { this.startDateEnabled = value; }
                public void setEndDateEnabled(boolean value) { this.endDateEnabled = value; }
                public void setInstructionEnabled(boolean value) { this.instructionEnabled = value; }
                public void setOrdinatorEnabled(boolean value) { this.ordinatorEnabled = value; }
                public void setSeponatorEnabled(boolean value) { this.seponatorEnabled = value; }
                public void setDosageIndex(int value) { this.dosageIndex = value; }
                public void setStartDateIndex(int value) { this.startDateIndex = value; }
                public void setEndDateIndex(int value) { this.endDateIndex = value; }
                public void setInstructionIndex(int value) { this.instructionIndex = value; }
                public void setOrdinatorIndex(int value) { this.ordinatorIndex = value; }
                public void setSeponatorIndex(int value) { this.seponatorIndex = value; }
                private void setDosageLabel(String value) { this.dosageLabel = value; }
                private void setStartDateLabel(String value) { this.startDateLabel = value; }
                private void setEndDateLabel(String value) { this.endDateLabel = value; }
                private void setInstructionLabel(String value) { this.instructionLabel = value; }
                private void setOrdinatorLabel(String value) { this.ordinatorLabel = value; }
                private void setSeponatorLabel(String value) { this.seponatorLabel = value; }
                public void setDrugVisible(boolean value) { this.drugVisible = value; }
                public void setDrugEnabled(boolean value) { this.drugEnabled = value; }
                public void setDrugIndex(int value) { this.drugIndex = value; }
                private void setDrugLabel(String value) { this.drugLabel = value; }
                private void createInteractors(int index){

        if (dosageVisible && dosageIndex == index){
          if (dosageEnabled)
                        dosageInteractorW = new
jinteractorcontrols.midp.MIDPToStringInteractor(this.form, this.dosageLabel, this.element.getDosage()) ;
                        else
                        dosageInteractorR = new
jinteractorcontrols.midp.MIDPStringToInteractor(this.form, this.dosageLabel, this.element.getDosage()) ;

                }

        if (startDateVisible && startDateIndex == index){
          if (startDateEnabled)
                        startDateInteractorW = new
jinteractorcontrols.midp.MIDPToDateInteractor(this.form, this.startDateLabel, this.element.getStartDate())
;
                        else
                        startDateInteractorR = new
jinteractorcontrols.midp.MIDPDateToInteractor(this.form, this.startDateLabel, this.element.getStartDate())
;
                }

        if (endDateVisible && endDateIndex == index){
          if (endDateEnabled)
                        endDateInteractorW = new
jinteractorcontrols.midp.MIDPToDateInteractor(this.form, this.endDateLabel, this.element.getEndDate()) ;
                        else
                        endDateInteractorR = new
jinteractorcontrols.midp.MIDPDateToInteractor(this.form, this.endDateLabel, this.element.getEndDate()) ;

                }
```

```
        if (instructionVisible && instructionIndex == index){
          if (instructionEnabled)
                        instructionInteractorW = new
jinteractorcontrols.midp.MIDPToStringInteractor(this.form, this.instructionLabel,
this.element.getInstruction()) ;
                        else
                        instructionInteractorR = new
jinteractorcontrols.midp.MIDPStringToInteractor(this.form, this.instructionLabel,
this.element.getInstruction()) ;
                }

        if (ordinatorVisible && ordinatorIndex == index){
          if (ordinatorEnabled)
                        ordinatorInteractorW = new
jinteractorcontrols.midp.MIDPToStringInteractor(this.form, this.ordinatorLabel,
this.element.getOrdinator()) ;
                        else
                        ordinatorInteractorR = new
jinteractorcontrols.midp.MIDPStringToInteractor(this.form, this.ordinatorLabel,
this.element.getOrdinator()) ;
                }

        if (seponatorVisible && seponatorIndex == index){
          if (seponatorEnabled)
                        seponatorInteractorW = new
jinteractorcontrols.midp.MIDPToStringInteractor(this.form, this.seponatorLabel,
this.element.getSeponator()) ;
                        else
                        seponatorInteractorR = new
jinteractorcontrols.midp.MIDPStringToInteractor(this.form, this.seponatorLabel,
this.element.getSeponator()) ;
                }



        if (drugVisible && drugIndex == index)
                drugInteractor = new jinteractorcontrols.midp.MIDPStringInteractor(this.form,
this.drugLabel, getDrugDisplayName((Object)this.element.getDrug()), 150, false) ;


                }
                private void updateElement(){

        if (dosageVisible && dosageEnabled)
                element.setDosage(dosageInteractorW.getValue());

        if (startDateVisible && startDateEnabled)
                element.setStartDate(startDateInteractorW.getValue());

        if (endDateVisible && endDateEnabled)
                element.setEndDate(endDateInteractorW.getValue());

        if (instructionVisible && instructionEnabled)
                element.setInstruction(instructionInteractorW.getValue());
```

```
            if (ordinatorVisible && ordinatorEnabled)
                        element.setOrdinator(ordinatorInteractorW.getValue());

            if (seponatorVisible && seponatorEnabled)
                        element.setSeponator(seponatorInteractorW.getValue());

                    }

                      public void activate(){
                              for (int i = 1; i < 1+1+1+1+1+1+1+1; i++)
                                      createInteractors(i-1);
                        Display.getDisplay(this.midlet).setCurrent(this.form);
                      }

                      public void setListener(jinteractorcontrols.IActionListener listener){
                        this.listener = listener;
                      }

                      public void commandAction(javax.microedition.lcdui.Command lcduiCmd, Displayable
displayable){
                        if (lcduiCmd.getCommandType() == javax.microedition.lcdui.Command.BACK){
                          listener.actionPerformed(this, jinteractorcontrols.IActionListener.BACK);
                        }
                        if (lcduiCmd.getCommandType() == javax.microedition.lcdui.Command.CANCEL){
                          listener.actionPerformed(this, jinteractorcontrols.IActionListener.CANCEL);
                        }
                        else
                          listener.actionPerformed(this, jinteractorcontrols.IActionListener.OK);
                      }

                    public String getDrugDisplayName(Object element){
        return ((medication.businessEntities.Drug)element).getCode().concat("
").concat(((medication.businessEntities.Drug)element).getName());
              }
                    public void initialize(){
        description = "New Medication";
        drugIndex = 0;
        startDateIndex = 1;
        endDateIndex = 2;
        dosageIndex = 3;
        instructionIndex = 4;
        ordinatorVisible = false;
        seponatorVisible = false;
        drugLabel = "Drug:";
        dosageLabel = "Dosage:";
        startDateLabel = "OrdinationDate:";
        endDateLabel = "SeponationDate:";
        instructionLabel = "Instruction:";
      }
}
```

Speech User Interface Client

The following section includes the generated source code for the speech user interface interactors.

AppointmentSelection.java

```
package medication.ui.phone;

public class AppointmentSelection extends
jinteractorcontrols.speech.ElementSelection {

        public
AppointmentSelection(jinteractorcontrols.speech.SpeechController
speechController){
          super(speechController);
        }


            public String getDisplayName(Object element){
          return
((medication.businessEntities.Appointment)element).getStartTime().toTimeStrin
g().concat("
").concat(((medication.businessEntities.Appointment)element).getPatient().get
LastName()).concat("
").concat(((medication.businessEntities.Appointment)element).getPatient().get
FirstName());
              }


            public void initialize(){
          description = "Select Appointment:";
        }
}
```

AtcLevelSelection.java

```
package medication.ui.phone;

import jinteractorcontrols.speech.SpeechController;

public class AtcLevelSelection extends
jinteractorcontrols.speech.HierarchicalSelection {

        public class ESelection extends
jinteractorcontrols.speech.ElementSelection {

          public ESelection(SpeechController speechController){
            super(speechController);
          }

            public String getDisplayName(Object element){
          return
((medication.businessEntities.AtcLevel)element).getCode().concat("
").concat(((medication.businessEntities.AtcLevel)element).getName());
               }


            public void initialize(){
```

```
          description = "Select ATC:";
        }


        }

        public AtcLevelSelection(SpeechController speechController){
            super(speechController);
                  initialize();
        }

              public void activate() {

activateSelection((medication.businessEntities.AtcLevel)this.rootElement);
              }

        public void activateSelection(medication.businessEntities.AtcLevel
element){
          ESelection selection = new ESelection(this.speechController);

          selection.setElements(element.getChildren());

          selection.setDescription(this.description);
          selection.setListener(new jinteractorcontrols.ActionListener() {
            public void actionPerformed(Object source, int type) {
              atcLevelSelectionPerformed(source, type);
            }
          });
           selection.activate();
        }

        public void atcLevelSelectionPerformed(Object source, int type){
          if (type == jinteractorcontrols.IActionListener.OK){
            jinteractorcontrols.ElementSelection selectionControl =
(jinteractorcontrols.ElementSelection) source;
            medication.businessEntities.AtcLevel element
                  = (medication.businessEntities.AtcLevel)
selectionControl.getSelectedElement();
            if (element.isComposite()) {
              activateSelection(element);
            }
            else {
              this.selectedElement = element;

this.selectionPerformed(jinteractorcontrols.IActionListener.OK);
            }
          }
          else if (type == jinteractorcontrols.IActionListener.BACK) {

this.selectionPerformed(jinteractorcontrols.IActionListener.BACK);
          }
          else if (type == jinteractorcontrols.IActionListener.CANCEL) {

this.selectionPerformed(jinteractorcontrols.IActionListener.CANCEL);
          }
          else if (type == jinteractorcontrols.IActionListener.EXIT) {
```

```
this.selectionPerformed(jinteractorcontrols.IActionListener.EXIT);
          }
        }
            public String getDisplayName(Object element){
          return
((medication.businessEntities.AtcLevel)element).getCode().concat("
").concat(((medication.businessEntities.AtcLevel)element).getName());
              }
            public void initialize(){
          description = "Select ATC:";
        }
}
```

DateSelection.java

```
package medication.ui.phone;

public class DateSelection extends
jinteractorcontrols.speech.ElementSelection {

        public DateSelection(jinteractorcontrols.speech.SpeechController
speechController){
          super(speechController);
        }


            public String getDisplayName(Object element){
          return ((jinteractorcontrols.Date)element).toDateString();
        }

            public void initialize(){
          description = "Select a Date:";
        }
}
```

MedicationSelection.java

```
package medication.ui.phone;

public class MedicationSelection extends
jinteractorcontrols.speech.ElementSelection {

        public
MedicationSelection(jinteractorcontrols.speech.SpeechController
speechController){
          super(speechController);
        }


            public String getDisplayName(Object element){
          return
((medication.businessEntities.Medication)element).getStartDate().toString().c
oncat("
").concat(((medication.businessEntities.Medication)element).getDrug().getName
());
```

```
            }

          public void initialize(){
        description = "Medications";
      }
}
```

MedicationView.java

```
package medication.ui.phone;

import java.util.ArrayList;

import jinteractorcontrols.speech.SpeechController;
import jinteractorcontrols.speech.StringToSpeechInteractor;
import jinteractorcontrols.speech.ToBooleanInteractor;
import jinteractorcontrols.speech.ToStringInteractor;
import jinteractorcontrols.speech.ToDateInteractor;
import jinteractorcontrols.speech.DateToSpeechInteractor;
import jinteractorcontrols.ActionListener;
import jinteractorcontrols.IActionListener;

public class MedicationView extends jinteractorcontrols.EntityView {

            private SpeechController speechController;
            private medication.businessEntities.Medication element;
            private boolean dosageVisible = true;
            private boolean startDateVisible = true;
            private boolean endDateVisible = true;
            private boolean instructionVisible = true;
            private boolean ordinatorVisible = true;
            private boolean seponatorVisible = true;
            private boolean dosageEnabled = true;
            private boolean startDateEnabled = true;
            private boolean endDateEnabled = true;
            private boolean instructionEnabled = true;
            private boolean ordinatorEnabled = true;
            private boolean seponatorEnabled = true;
            private int dosageIndex = 0;
            private int startDateIndex = 0;
            private int endDateIndex = 0;
            private int instructionIndex = 0;
            private int ordinatorIndex = 0;
            private int seponatorIndex = 0;
            private String dosageLabel = "dosage";
            private String startDateLabel = "startDate";
            private String endDateLabel = "endDate";
            private String instructionLabel = "instruction";
            private String ordinatorLabel = "ordinator";
            private String seponatorLabel = "seponator";
            private jinteractorcontrols.speech.StringToSpeechInteractor
dosageInteractor;
            private jinteractorcontrols.speech.DateToSpeechInteractor
startDateInteractor;
            private jinteractorcontrols.speech.DateToSpeechInteractor
endDateInteractor;
```

```
            private jinteractorcontrols.speech.StringToSpeechInteractor
instructionInteractor;
            private jinteractorcontrols.speech.StringToSpeechInteractor
ordinatorInteractor;
            private jinteractorcontrols.speech.StringToSpeechInteractor
seponatorInteractor;
            private boolean drugVisible = true;
            private boolean drugEnabled = true;
            private int drugIndex = 0;
            private String drugLabel = "drug";
            private jinteractorcontrols.speech.StringToSpeechInteractor
drugInteractor;
            private ArrayList activatedControls;
            private int currentIndex;
            private int maxIndex = 1+1+1+1+1+1+1+1;

        public MedicationView(SpeechController speechController) {
          this.speechController = speechController;
          initialize();
        }

            public void setMedication(medication.businessEntities.Medication
value) {this.element = value;}
            public medication.businessEntities.Medication getMedication() {
return this.element;}
            public void setDosageVisible(boolean value) { this.dosageVisible
= value; }
            public void setStartDateVisible(boolean value) {
this.startDateVisible = value; }
            public void setEndDateVisible(boolean value) {
this.endDateVisible = value; }
            public void setInstructionVisible(boolean value) {
this.instructionVisible = value; }
            public void setOrdinatorVisible(boolean value) {
this.ordinatorVisible = value; }
            public void setSeponatorVisible(boolean value) {
this.seponatorVisible = value; }



            public void setDosageEnabled(boolean value) { this.dosageEnabled
= value; }
            public void setStartDateEnabled(boolean value) {
this.startDateEnabled = value; }
            public void setEndDateEnabled(boolean value) {
this.endDateEnabled = value; }
            public void setInstructionEnabled(boolean value) {
this.instructionEnabled = value; }
            public void setOrdinatorEnabled(boolean value) {
this.ordinatorEnabled = value; }
            public void setSeponatorEnabled(boolean value) {
this.seponatorEnabled = value; }
            public void setDosageIndex(int value) { this.dosageIndex = value;
}
            public void setStartDateIndex(int value) { this.startDateIndex =
value; }
```

```
            public void setEndDateIndex(int value) { this.endDateIndex =
value; }
            public void setInstructionIndex(int value) {
this.instructionIndex = value; }
            public void setOrdinatorIndex(int value) { this.ordinatorIndex =
value; }
            public void setSeponatorIndex(int value) { this.seponatorIndex =
value; }
            private void setDosageLabel(String value) { this.dosageLabel =
value; }
            private void setStartDateLabel(String value) {
this.startDateLabel = value; }
            private void setEndDateLabel(String value) { this.endDateLabel =
value; }
            private void setInstructionLabel(String value) {
this.instructionLabel = value; }
            private void setOrdinatorLabel(String value) {
this.ordinatorLabel = value; }
            private void setSeponatorLabel(String value) {
this.seponatorLabel = value; }
            public void setDrugVisible(boolean value) { this.drugVisible =
value; }
            public void setDrugEnabled(boolean value) { this.drugEnabled =
value; }
            public void setDrugIndex(int value) { this.drugIndex = value; }
            private void setDrugLabel(String value) { this.drugLabel = value;
}

              public void activate(){
                  this.activatedControls = new ArrayList();
                  currentIndex = -1;
                  activateNext();
              }

              public void activateNext(){
                  if (currentIndex < maxIndex){
                        currentIndex++;


                        if (this.dosageVisible && this.dosageIndex ==
this.currentIndex && !this.activatedControls.contains("dosage"))
                        {
                              activateDosage();
                              this.activatedControls.add("dosage");
                              return;
                        }

                        if (this.startDateVisible && this.startDateIndex ==
this.currentIndex && !this.activatedControls.contains("startDate"))
                        {
                              activateStartDate();
                              this.activatedControls.add("startDate");
                              return;
                        }

                        if (this.endDateVisible && this.endDateIndex ==
this.currentIndex && !this.activatedControls.contains("endDate"))
```

```
                        {
                              activateEndDate();
                              this.activatedControls.add("endDate");
                              return;
                        }

                        if (this.instructionVisible && this.instructionIndex
== this.currentIndex && !this.activatedControls.contains("instruction"))
                        {
                              activateInstruction();
                              this.activatedControls.add("instruction");
                              return;
                        }

                        if (this.ordinatorVisible && this.ordinatorIndex ==
this.currentIndex && !this.activatedControls.contains("ordinator"))
                        {
                              activateOrdinator();
                              this.activatedControls.add("ordinator");
                              return;
                        }

                        if (this.seponatorVisible && this.seponatorIndex ==
this.currentIndex && !this.activatedControls.contains("seponator"))
                        {
                              activateSeponator();
                              this.activatedControls.add("seponator");
                              return;
                        }


                        if (this.drugVisible && this.drugIndex ==
this.currentIndex && !this.activatedControls.contains("drug"))
                        {
                              activateDrug();
                              this.activatedControls.add("drug");
                              return;
                        }

                        activateNext();
                  }
                  else{
                        finished();
                  }

            }


            public void initialize(){
          description = "Medication";
          drugIndex = 0;
          startDateIndex = 1;
          endDateIndex = 2;
          dosageIndex = 3;
          instructionIndex = 4;
          ordinatorVisible = false;
          seponatorVisible = false;
```

```
          dosageEnabled = false;
          startDateEnabled = false;
          instructionEnabled = false;
          drugLabel = "Drug:";
          dosageLabel = "Dosage:";
          startDateLabel = "OrdinationDate:";
          endDateLabel = "SeponationDate:";
          instructionLabel = "Instruction:";
        }


            public String getDrugDisplayName(Object element){
          return
((medication.businessEntities.Drug)element).getCode().concat("
").concat(((medication.businessEntities.Drug)element).getName());
                }




            private void activateDosage(){
            if (dosageVisible){
                  String description = "Attribute " + dosageLabel + "
contains the value: ";
                  String emptyDescription = "Attribute " + dosageLabel + "
dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);


                  if (this.element.getDosage() == null ||
this.element.getDosage() == ""){


                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);
                        valueStringTo.setValue(this.element.getDosage());
                  }

                  if (dosageEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
```

```
                                    ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                    boolean value = doChange.getValue();
                                    if (value){
                                        getDosage();
                                    } else {
                                        activateNext();
                                    }
                                }
                          }
                          });
                          doChange.activate();
                    }else{
                          activateNext();
                    }
              }else{
                    activateNext();
              }
        }

        private void getDosage(){
              ToStringInteractor interactor = new
ToStringInteractor(this.speechController);
              interactor.setListener(new ActionListener() {
              public void actionPerformed(Object source, int type) {
                dosageChanged(source, type);
              }
              });
              interactor.activate();

        }

        public void dosageChanged(Object source, int type){
          if (type == IActionListener.OK){
              ToStringInteractor interactor = (ToStringInteractor)source;
              this.element.setDosage(interactor.getValue());
              activateNext();
          }
        }

              private void activateStartDate(){
              if (startDateVisible){
                    String description = "Attribute " + startDateLabel + "
contains the value: ";
                    String emptyDescription = "Attribute " + startDateLabel + "
dosage contains no value. ";
                    String changeDescription = "Do you want to change this
value? ";

                    StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                    DateToSpeechInteractor valueStringTo = new
DateToSpeechInteractor(this.speechController);



                    if (this.element.getStartDate() == null){
```

```
                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);
                        valueStringTo.setValue(this.element.getStartDate());
                  }

                  if (startDateEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
                                ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                boolean value = doChange.getValue();
                                if (value){
                                    getStartDate();
                                } else {
                                    activateNext();
                                }
                              }
                        }
                        });
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
                  activateNext();
            }
      }

      private void getStartDate(){
            ToDateInteractor interactor = new
ToDateInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              startDateChanged(source, type);
            }
            });
            interactor.activate();

      }

      public void startDateChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToDateInteractor interactor = (ToDateInteractor)source;
            this.element.setStartDate(interactor.getValue());
            activateNext();
        }
      }
```

```
            private void activateEndDate(){
            if (endDateVisible){
                  String description = "Attribute " + endDateLabel + "
contains the value: ";
                  String emptyDescription = "Attribute " + endDateLabel + "
dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  DateToSpeechInteractor valueStringTo = new
DateToSpeechInteractor(this.speechController);



                  if (this.element.getEndDate() == null){

                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);
                        valueStringTo.setValue(this.element.getEndDate());
                  }

                  if (endDateEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
                                ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                boolean value = doChange.getValue();
                                if (value){
                                    getEndDate();
                                } else {
                                    activateNext();
                                }
                              }
                        }
                        });
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
                  activateNext();
            }
      }

      private void getEndDate(){
```

```
            ToDateInteractor interactor = new
ToDateInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              endDateChanged(source, type);
            }
            });
            interactor.activate();

      }

      public void endDateChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToDateInteractor interactor = (ToDateInteractor)source;
            this.element.setEndDate(interactor.getValue());
            activateNext();
        }
      }

            private void activateInstruction(){
            if (instructionVisible){
                  String description = "Attribute " + instructionLabel + "
contains the value: ";
                  String emptyDescription = "Attribute " + instructionLabel +
" dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);


                  if (this.element.getInstruction() == null ||
this.element.getInstruction() == ""){


                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);

      valueStringTo.setValue(this.element.getInstruction());
                  }

                  if (instructionEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
```

```
                                    ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                    boolean value = doChange.getValue();
                                    if (value){
                                        getInstruction();
                                    } else {
                                        activateNext();
                                    }
                                }
                            }
                            });
                            doChange.activate();
                      }else{
                            activateNext();
                      }
                }else{
                      activateNext();
                }
          }

          private void getInstruction(){
                ToStringInteractor interactor = new
ToStringInteractor(this.speechController);
                interactor.setListener(new ActionListener() {
                public void actionPerformed(Object source, int type) {
                  instructionChanged(source, type);
                }
                });
                interactor.activate();

          }

          public void instructionChanged(Object source, int type){
            if (type == IActionListener.OK){
                ToStringInteractor interactor = (ToStringInteractor)source;
                this.element.setInstruction(interactor.getValue());
                activateNext();
            }
          }

                private void activateOrdinator(){
                if (ordinatorVisible){
                      String description = "Attribute " + ordinatorLabel + "
contains the value: ";
                      String emptyDescription = "Attribute " + ordinatorLabel + "
dosage contains no value. ";
                      String changeDescription = "Do you want to change this
value? ";

                      StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                      StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);


                      if (this.element.getOrdinator() == null ||
this.element.getOrdinator() == ""){
```

```
                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);
                        valueStringTo.setValue(this.element.getOrdinator());
                  }

                  if (ordinatorEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
                                ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                boolean value = doChange.getValue();
                                if (value){
                                    getOrdinator();
                                } else {
                                    activateNext();
                                }
                              }
                        }
                        });
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
                  activateNext();
            }
      }

      private void getOrdinator(){
            ToStringInteractor interactor = new
ToStringInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              ordinatorChanged(source, type);
            }
            });
            interactor.activate();

      }

      public void ordinatorChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToStringInteractor interactor = (ToStringInteractor)source;
            this.element.setOrdinator(interactor.getValue());
            activateNext();
        }
```

```
	}

		private void activateSeponator(){
		if (seponatorVisible){
			String description = "Attribute " + seponatorLabel + "
contains the value: ";
			String emptyDescription = "Attribute " + seponatorLabel + "
dosage contains no value. ";
			String changeDescription = "Do you want to change this
value? ";

			StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
			StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);


			if (this.element.getSeponator() == null ||
this.element.getSeponator() == ""){


				descriptionStringTo.setValue(emptyDescription);
			}else{
				descriptionStringTo.setValue(description);
				valueStringTo.setValue(this.element.getSeponator());
			}

			if (seponatorEnabled){
				StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
				actionStringTo.setValue(changeDescription);

				ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
				doChange.setListener(new ActionListener() {
				public void actionPerformed(Object source, int type)
{
					if (type == IActionListener.OK){
					  ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
					  boolean value = doChange.getValue();
					  if (value){
					      getSeponator();
					  } else {
					      activateNext();
					  }
					}
				}
				});
				doChange.activate();
			}else{
				activateNext();
			}
		}else{
			activateNext();
		}
	}
```

```
      private void getSeponator(){
            ToStringInteractor interactor = new
ToStringInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              seponatorChanged(source, type);
            }
            });
            interactor.activate();

      }

      public void seponatorChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToStringInteractor interactor = (ToStringInteractor)source;
            this.element.setSeponator(interactor.getValue());
            activateNext();
        }
      }

            private void activateDrug(){
            if (drugVisible){
                  String description = "Attribute " + drugLabel + " contains
the value: ";
                  String emptyDescription = "Attribute " + drugLabel + "
dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);
                  if (getDrugDisplayName(this.element) == null ||
getDrugDisplayName(this.element) == ""){
                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);

      valueStringTo.setValue(getDrugDisplayName(this.element));
                  }

            }else{
                  activateNext();
            }
      }

      private void finished(){
            this.selectionPerformed(IActionListener.OK);
      }
}
```

OrdinateView.java

```
package medication.ui.phone;
```

```
import java.util.ArrayList;

import jinteractorcontrols.speech.SpeechController;
import jinteractorcontrols.speech.StringToSpeechInteractor;
import jinteractorcontrols.speech.ToBooleanInteractor;
import jinteractorcontrols.speech.ToStringInteractor;
import jinteractorcontrols.speech.ToDateInteractor;
import jinteractorcontrols.speech.DateToSpeechInteractor;
import jinteractorcontrols.ActionListener;
import jinteractorcontrols.IActionListener;

public class OrdinateView extends jinteractorcontrols.EntityView {

            private SpeechController speechController;

            private medication.businessEntities.Medication element;
            private boolean dosageVisible = true;
            private boolean startDateVisible = true;
            private boolean endDateVisible = true;
            private boolean instructionVisible = true;
            private boolean ordinatorVisible = true;
            private boolean seponatorVisible = true;
            private boolean dosageEnabled = true;
            private boolean startDateEnabled = true;
            private boolean endDateEnabled = true;
            private boolean instructionEnabled = true;
            private boolean ordinatorEnabled = true;
            private boolean seponatorEnabled = true;
            private int dosageIndex = 0;
            private int startDateIndex = 0;
            private int endDateIndex = 0;
            private int instructionIndex = 0;
            private int ordinatorIndex = 0;
            private int seponatorIndex = 0;
            private String dosageLabel = "dosage";
            private String startDateLabel = "startDate";
            private String endDateLabel = "endDate";
            private String instructionLabel = "instruction";
            private String ordinatorLabel = "ordinator";
            private String seponatorLabel = "seponator";
            private jinteractorcontrols.speech.StringToSpeechInteractor
dosageInteractor;
            private jinteractorcontrols.speech.DateToSpeechInteractor
startDateInteractor;
            private jinteractorcontrols.speech.DateToSpeechInteractor
endDateInteractor;
            private jinteractorcontrols.speech.StringToSpeechInteractor
instructionInteractor;
            private jinteractorcontrols.speech.StringToSpeechInteractor
ordinatorInteractor;
            private jinteractorcontrols.speech.StringToSpeechInteractor
seponatorInteractor;
            private boolean drugVisible = true;
            private boolean drugEnabled = true;
            private int drugIndex = 0;
            private String drugLabel = "drug";
```

```
            private jinteractorcontrols.speech.StringToSpeechInteractor
drugInteractor;
            private ArrayList activatedControls;
            private int currentIndex;
            private int maxIndex = 1+1+1+1+1+1+1+1;

        public OrdinateView(SpeechController speechController) {
          this.speechController = speechController;
          initialize();
        }

            public void setMedication(medication.businessEntities.Medication
value) {this.element = value;}
            public medication.businessEntities.Medication getMedication() {
return this.element;}
            public void setDosageVisible(boolean value) { this.dosageVisible
= value; }
            public void setStartDateVisible(boolean value) {
this.startDateVisible = value; }
            public void setEndDateVisible(boolean value) {
this.endDateVisible = value; }
            public void setInstructionVisible(boolean value) {
this.instructionVisible = value; }
            public void setOrdinatorVisible(boolean value) {
this.ordinatorVisible = value; }
            public void setSeponatorVisible(boolean value) {
this.seponatorVisible = value; }
            public void setDosageEnabled(boolean value) { this.dosageEnabled
= value; }
            public void setStartDateEnabled(boolean value) {
this.startDateEnabled = value; }
            public void setEndDateEnabled(boolean value) {
this.endDateEnabled = value; }
            public void setInstructionEnabled(boolean value) {
this.instructionEnabled = value; }
            public void setOrdinatorEnabled(boolean value) {
this.ordinatorEnabled = value; }
            public void setSeponatorEnabled(boolean value) {
this.seponatorEnabled = value; }
            public void setDosageIndex(int value) { this.dosageIndex = value;
}
            public void setStartDateIndex(int value) { this.startDateIndex =
value; }
            public void setEndDateIndex(int value) { this.endDateIndex =
value; }
            public void setInstructionIndex(int value) {
this.instructionIndex = value; }
            public void setOrdinatorIndex(int value) { this.ordinatorIndex =
value; }
            public void setSeponatorIndex(int value) { this.seponatorIndex =
value; }
            private void setDosageLabel(String value) { this.dosageLabel =
value; }
            private void setStartDateLabel(String value) {
this.startDateLabel = value; }
            private void setEndDateLabel(String value) { this.endDateLabel =
value; }
```

```
            private void setInstructionLabel(String value) {
this.instructionLabel = value; }
            private void setOrdinatorLabel(String value) {
this.ordinatorLabel = value; }
            private void setSeponatorLabel(String value) {
this.seponatorLabel = value; }
            public void setDrugVisible(boolean value) { this.drugVisible =
value; }
            public void setDrugEnabled(boolean value) { this.drugEnabled =
value; }
            public void setDrugIndex(int value) { this.drugIndex = value; }
            private void setDrugLabel(String value) { this.drugLabel = value;
}



              public void activate(){
                  this.activatedControls = new ArrayList();
                  currentIndex = -1;
                  activateNext();
              }

              public void activateNext(){
                  if (currentIndex < maxIndex){
                        currentIndex++;


                        if (this.dosageVisible && this.dosageIndex ==
this.currentIndex && !this.activatedControls.contains("dosage"))
                        {
                              activateDosage();
                              this.activatedControls.add("dosage");
                              return;
                        }

                        if (this.startDateVisible && this.startDateIndex ==
this.currentIndex && !this.activatedControls.contains("startDate"))
                        {
                              activateStartDate();
                              this.activatedControls.add("startDate");
                              return;
                        }

                        if (this.endDateVisible && this.endDateIndex ==
this.currentIndex && !this.activatedControls.contains("endDate"))
                        {
                              activateEndDate();
                              this.activatedControls.add("endDate");
                              return;
                        }

                        if (this.instructionVisible && this.instructionIndex
== this.currentIndex && !this.activatedControls.contains("instruction"))
                        {
                              activateInstruction();
                              this.activatedControls.add("instruction");
                              return;
```

```
                        }

                        if (this.ordinatorVisible && this.ordinatorIndex ==
this.currentIndex && !this.activatedControls.contains("ordinator"))
                        {
                              activateOrdinator();
                              this.activatedControls.add("ordinator");
                              return;
                        }

                        if (this.seponatorVisible && this.seponatorIndex ==
this.currentIndex && !this.activatedControls.contains("seponator"))
                        {
                              activateSeponator();
                              this.activatedControls.add("seponator");
                              return;
                        }


                        if (this.drugVisible && this.drugIndex ==
this.currentIndex && !this.activatedControls.contains("drug"))
                        {
                              activateDrug();
                              this.activatedControls.add("drug");
                              return;
                        }

                        activateNext();
                  }
                  else{
                        finished();
                  }

              }


            public String getDrugDisplayName(Object element){
           return
((medication.businessEntities.Drug)element).getCode().concat("
").concat(((medication.businessEntities.Drug)element).getName());
                }


            public void initialize(){
           description = "New Medication";
           drugIndex = 0;
           startDateIndex = 1;
           endDateIndex = 2;
           dosageIndex = 3;
           instructionIndex = 4;
           ordinatorVisible = false;
           seponatorVisible = false;
           drugLabel = "Drug:";
           dosageLabel = "Dosage:";
           startDateLabel = "OrdinationDate:";
           endDateLabel = "SeponationDate:";
           instructionLabel = "Instruction:";
```

```
        }



            private void activateDosage(){
            if (dosageVisible){
                  String description = "Attribute " + dosageLabel + "
contains the value: ";
                  String emptyDescription = "Attribute " + dosageLabel + "
dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);


                  if (this.element.getDosage() == null ||
this.element.getDosage() == ""){


                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);
                        valueStringTo.setValue(this.element.getDosage());
                  }

                  if (dosageEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
                                ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                boolean value = doChange.getValue();
                                if (value){
                                    getDosage();
                                } else {
                                    activateNext();
                                }
                              }
                        }
                        });
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
```

```
                activateNext();
            }
      }

      private void getDosage(){
            ToStringInteractor interactor = new
ToStringInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              dosageChanged(source, type);
            }
            });
            interactor.activate();

      }

      public void dosageChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToStringInteractor interactor = (ToStringInteractor)source;
            this.element.setDosage(interactor.getValue());
            activateNext();
        }
      }

            private void activateStartDate(){
            if (startDateVisible){
                  String description = "Attribute " + startDateLabel + "
contains the value: ";
                  String emptyDescription = "Attribute " + startDateLabel + "
dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  DateToSpeechInteractor valueStringTo = new
DateToSpeechInteractor(this.speechController);



                  if (this.element.getStartDate() == null){

                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);
                        valueStringTo.setValue(this.element.getStartDate());
                  }

                  if (startDateEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
```

```
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
                                ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                boolean value = doChange.getValue();
                                if (value){
                                    getStartDate();
                                } else {
                                    activateNext();
                                }
                              }
                        }
                        });
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
                  activateNext();
            }
      }

      private void getStartDate(){
            ToDateInteractor interactor = new
ToDateInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              startDateChanged(source, type);
            }
            });
            interactor.activate();

      }

      public void startDateChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToDateInteractor interactor = (ToDateInteractor)source;
            this.element.setStartDate(interactor.getValue());
            activateNext();
        }
      }

            private void activateEndDate(){
            if (endDateVisible){
                  String description = "Attribute " + endDateLabel + "
contains the value: ";
                  String emptyDescription = "Attribute " + endDateLabel + "
dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  DateToSpeechInteractor valueStringTo = new
DateToSpeechInteractor(this.speechController);
```

```
                  if (this.element.getEndDate() == null){

                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);
                        valueStringTo.setValue(this.element.getEndDate());
                  }

                  if (endDateEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
                                ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                boolean value = doChange.getValue();
                                if (value){
                                    getEndDate();
                                } else {
                                    activateNext();
                                }
                              }
                        }
                        });
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
                  activateNext();
            }
      }

      private void getEndDate(){
            ToDateInteractor interactor = new
ToDateInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              endDateChanged(source, type);
            }
            });
            interactor.activate();

      }

      public void endDateChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToDateInteractor interactor = (ToDateInteractor)source;
            this.element.setEndDate(interactor.getValue());
```

```
            activateNext();
          }
      }

            private void activateInstruction(){
            if (instructionVisible){
                  String description = "Attribute " + instructionLabel + "
contains the value: ";
                  String emptyDescription = "Attribute " + instructionLabel +
" dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);


                  if (this.element.getInstruction() == null ||
this.element.getInstruction() == ""){


                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);

      valueStringTo.setValue(this.element.getInstruction());
                  }

                  if (instructionEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
                                ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                boolean value = doChange.getValue();
                                if (value){
                                    getInstruction();
                                } else {
                                    activateNext();
                                }
                              }
                        }
                        });
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
```

```
                activateNext();
            }
      }

      private void getInstruction(){
            ToStringInteractor interactor = new
ToStringInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              instructionChanged(source, type);
            }
            });
            interactor.activate();

      }

      public void instructionChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToStringInteractor interactor = (ToStringInteractor)source;
            this.element.setInstruction(interactor.getValue());
            activateNext();
        }
      }

            private void activateOrdinator(){
            if (ordinatorVisible){
                  String description = "Attribute " + ordinatorLabel + "
contains the value: ";
                  String emptyDescription = "Attribute " + ordinatorLabel + "
dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);


                  if (this.element.getOrdinator() == null ||
this.element.getOrdinator() == ""){


                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);
                        valueStringTo.setValue(this.element.getOrdinator());
                  }

                  if (ordinatorEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
```

```
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
                                ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                boolean value = doChange.getValue();
                                if (value){
                                    getOrdinator();
                                } else {
                                    activateNext();
                                }
                              }
                        }
                        });
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
                  activateNext();
            }
      }

      private void getOrdinator(){
            ToStringInteractor interactor = new
ToStringInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              ordinatorChanged(source, type);
            }
            });
            interactor.activate();

      }

      public void ordinatorChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToStringInteractor interactor = (ToStringInteractor)source;
            this.element.setOrdinator(interactor.getValue());
            activateNext();
        }
      }

            private void activateSeponator(){
            if (seponatorVisible){
                  String description = "Attribute " + seponatorLabel + "
contains the value: ";
                  String emptyDescription = "Attribute " + seponatorLabel + "
dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);
```

```
                  if (this.element.getSeponator() == null ||
this.element.getSeponator() == ""){


                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);
                        valueStringTo.setValue(this.element.getSeponator());
                  }

                  if (seponatorEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
                                ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                boolean value = doChange.getValue();
                                if (value){
                                    getSeponator();
                                } else {
                                    activateNext();
                                }
                              }
                        }
                        });
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
                  activateNext();
            }
      }

      private void getSeponator(){
            ToStringInteractor interactor = new
ToStringInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              seponatorChanged(source, type);
            }
            });
            interactor.activate();

      }

      public void seponatorChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToStringInteractor interactor = (ToStringInteractor)source;
```

```
            this.element.setSeponator(interactor.getValue());
            activateNext();
        }
      }



            private void activateDrug(){
            if (drugVisible){
                  String description = "Attribute " + drugLabel + " contains
the value: ";
                  String emptyDescription = "Attribute " + drugLabel + "
dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);
                  if (getDrugDisplayName(this.element) == null ||
getDrugDisplayName(this.element) == ""){
                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);

      valueStringTo.setValue(getDrugDisplayName(this.element));
                  }

            }else{
                  activateNext();
            }
      }

      private void finished(){
            this.selectionPerformed(IActionListener.OK);
      }
}
```

PatientRetrieval.java

```
package medication.ui.phone;

import jinteractorcontrols.speech.SpeechController;

public class PatientRetrieval extends
jinteractorcontrols.speech.InteractorSelection {


        public PatientRetrieval(SpeechController speechController){
          super(speechController);

          initialize();

        }
```

```
            public void initialize(){
          ;
        }
}
```

PatientRetrievalByAppointmentCommand.java

```
package medication.ui.phone;

public class PatientRetrievalByAppointmentCommand extends
jinteractorcontrols.Command {

        public PatientRetrievalByAppointmentCommand(){
          initialize();
        }

            public void initialize(){
          name = "Appointments";
        }
}
```

PatientSearchCommand.java

```
package medication.ui.phone;

public class PatientSearchCommand extends jinteractorcontrols.Command {

        public PatientSearchCommand(){
          initialize();
        }

            public void initialize(){
          name = "Patient Search";
        }
}
```

PatientSearchCriteriaView.java

```
package medication.ui.phone;

import java.util.ArrayList;
import jinteractorcontrols.speech.SpeechController;
import jinteractorcontrols.speech.StringToSpeechInteractor;
import jinteractorcontrols.speech.ToBooleanInteractor;
import jinteractorcontrols.speech.ToStringInteractor;
import jinteractorcontrols.speech.ToDateInteractor;
import jinteractorcontrols.speech.DateToSpeechInteractor;
import jinteractorcontrols.ActionListener;
import jinteractorcontrols.IActionListener;

public class PatientSearchCriteriaView extends jinteractorcontrols.EntityView
{

            private SpeechController speechController;
```

```
            private medication.businessEntities.PatientSearchCriteria
element;
            private boolean searchStringVisible = true;
            private boolean searchStringEnabled = true;
            private int searchStringIndex = 0;
            private String searchStringLabel = "searchString";
            private jinteractorcontrols.speech.StringToSpeechInteractor
searchStringInteractor;
            private boolean patientSearchTypeVisible = true;
            private boolean patientSearchTypeEnabled = true;
            private int patientSearchTypeIndex = 0;
            private String patientSearchTypeLabel = "patientSearchType";
            private jinteractorcontrols.speech.StringToSpeechInteractor
patientSearchTypeInteractor;
            private ArrayList activatedControls;
            private int currentIndex;
            private int maxIndex = 1+1+1;

        public PatientSearchCriteriaView(SpeechController speechController) {
          this.speechController = speechController;
          initialize();
        }

            public void
setPatientSearchCriteria(medication.businessEntities.PatientSearchCriteria
value) {this.element = value;}
            public medication.businessEntities.PatientSearchCriteria
getPatientSearchCriteria() { return this.element;}
            public void setSearchStringVisible(boolean value) {
this.searchStringVisible = value; }
            public void setSearchStringEnabled(boolean value) {
this.searchStringEnabled = value; }
            public void setSearchStringIndex(int value) {
this.searchStringIndex = value; }
            private void setSearchStringLabel(String value) {
this.searchStringLabel = value; }
            public void setPatientSearchTypeVisible(boolean value) {
this.patientSearchTypeVisible = value; }
            public void setPatientSearchTypeEnabled(boolean value) {
this.patientSearchTypeEnabled = value; }
            public void setPatientSearchTypeIndex(int value) {
this.patientSearchTypeIndex = value; }
            private void setPatientSearchTypeLabel(String value) {
this.patientSearchTypeLabel = value; }



              public void activate(){
                  this.activatedControls = new ArrayList();
                  currentIndex = -1;
                  activateNext();
              }

              public void activateNext(){
                  if (currentIndex < maxIndex){
                        currentIndex++;
```

```
                        if (this.searchStringVisible &&
this.searchStringIndex == this.currentIndex &&
!this.activatedControls.contains("searchString"))
                        {
                              activateSearchString();
                              this.activatedControls.add("searchString");
                              return;
                        }


                        if (this.patientSearchTypeVisible &&
this.patientSearchTypeIndex == this.currentIndex &&
!this.activatedControls.contains("patientSearchType"))
                        {
                              activatePatientSearchType();

      this.activatedControls.add("patientSearchType");
                              return;
                        }

                        activateNext();
                  }
                  else{
                        finished();
                  }

            }


            public void initialize(){
          description = "Set Query String:";
          patientSearchTypeVisible = false;
          searchStringEnabled = true;
          searchStringVisible = true;
          searchStringIndex = 0;
          searchStringLabel = "Query:";
        }


            public String getPatientSearchTypeDisplayName(Object element){
          return
((medication.businessEntities.PatientSearchType)element).getName();
        }

            private void activateSearchString(){
            if (searchStringVisible){
                  String description = "Attribute " + searchStringLabel + "
contains the value: ";
                  String emptyDescription = "Attribute " + searchStringLabel
+ " dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
```

```
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);


                  if (this.element.getSearchString() == null ||
this.element.getSearchString() == ""){


                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);

      valueStringTo.setValue(this.element.getSearchString());
                  }

                  if (searchStringEnabled){
                        StringToSpeechInteractor actionStringTo = new
StringToSpeechInteractor(this.speechController);
                        actionStringTo.setValue(changeDescription);

                        ToBooleanInteractor doChange = new
ToBooleanInteractor(this.speechController);
                        doChange.setListener(new ActionListener() {
                        public void actionPerformed(Object source, int type)
{
                              if (type == IActionListener.OK){
                                ToBooleanInteractor doChange =
(ToBooleanInteractor)source;
                                boolean value = doChange.getValue();
                                if (value){
                                    getSearchString();
                                } else {
                                    activateNext();
                                }
                              }
                        }
                        });
                        doChange.activate();
                  }else{
                        activateNext();
                  }
            }else{
                  activateNext();
            }
      }

      private void getSearchString(){
            ToStringInteractor interactor = new
ToStringInteractor(this.speechController);
            interactor.setListener(new ActionListener() {
            public void actionPerformed(Object source, int type) {
              searchStringChanged(source, type);
            }
            });
            interactor.activate();

      }
```

```
      public void searchStringChanged(Object source, int type){
        if (type == IActionListener.OK){
            ToStringInteractor interactor = (ToStringInteractor)source;
            this.element.setSearchString(interactor.getValue());
            activateNext();
        }
      }

            private void activatePatientSearchType(){
            if (patientSearchTypeVisible){
                  String description = "Attribute " + patientSearchTypeLabel
+ " contains the value: ";
                  String emptyDescription = "Attribute " +
patientSearchTypeLabel + " dosage contains no value. ";
                  String changeDescription = "Do you want to change this
value? ";

                  StringToSpeechInteractor descriptionStringTo = new
StringToSpeechInteractor(this.speechController);
                  StringToSpeechInteractor valueStringTo = new
StringToSpeechInteractor(this.speechController);
                  if (getPatientSearchTypeDisplayName(this.element) == null
|| getPatientSearchTypeDisplayName(this.element) == ""){
                        descriptionStringTo.setValue(emptyDescription);
                  }else{
                        descriptionStringTo.setValue(description);

      valueStringTo.setValue(getPatientSearchTypeDisplayName(this.element));
                  }

            }else{
                  activateNext();
            }
      }

      private void finished(){
            this.selectionPerformed(IActionListener.OK);
      }
}
```

PatientSearchTypeSelection.java

```
package medication.ui.phone;

public class PatientSearchTypeSelection extends
jinteractorcontrols.speech.ElementSelection {

        public
PatientSearchTypeSelection(jinteractorcontrols.speech.SpeechController
speechController){
          super(speechController);
        }


            public String getDisplayName(Object element){
```

```
          return
((medication.businessEntities.PatientSearchType)element).getName();
        }

            public void initialize(){
          description = "Select a Search Type:";
        }
}
```

PatientSelection.java

```
package medication.ui.phone;

public class PatientSelection extends
jinteractorcontrols.speech.ElementSelection {

        public PatientSelection(jinteractorcontrols.speech.SpeechController
speechController){
          super(speechController);
        }


            public String getDisplayName(Object element){
          return
((medication.businessEntities.Patient)element).getFirstName().concat("
").concat(((medication.businessEntities.Patient)element).getMiddleName()).con
cat("
").concat(((medication.businessEntities.Patient)element).getLastName());
              }


            public void initialize(){
          description = "Select Patient:";
        }
}
```

PatientView.java

```
package medication.ui.phone;

import jinteractorcontrols.speech.SpeechController;

public class PatientView extends
jinteractorcontrols.speech.InteractorSelection {


        public PatientView(SpeechController speechController){
          super(speechController);

          initialize();

        }

            public void initialize(){
          ;
        }
```

```
}
```

MedicationSelectionViewCommand.java

```
package medication.ui.phone;

public class MedicationsSelectionViewCommand extends
jinteractorcontrols.Command {

        public MedicationsSelectionViewCommand(){
          initialize();
        }

            public void initialize(){
          name = "View";
        }

}
```

OrdinateCommand.java

```
package medication.ui.phone;

public class OrdinateCommand extends jinteractorcontrols.Command {

        public OrdinateCommand(){
          initialize();
        }

            public void initialize(){
          name = "Ordinate";
        }
}
```