2009 Fourth South-East European Workshop on Formal Methods

Formal Visual Modelling of Human Agents in Service Oriented Systems

Adwoa Donyina
Department of Computer Science
University of Leicester
Leicester, United Kingdom
Email: add7@leicester.ac.uk

Reiko Heckel
Department of Computer Science
University of Leicester
Leicester, United Kingdom
Email: reiko@mcs.le.ac.uk

***Abstract*—When the level of granularity of services approaches that of business activities, humans become part of a service-oriented system not just as users but as providers of services. A model of such a system has to take into account the characteristics of human actors as service providers. Conversely, in the world of agent-based systems, software components have been attributed with human properties such as reactivity, autonomy and pro-activity. We believe that modelling techniques developed for software agents are a valid starting point for specifying human agents in service-oriented systems (HASOS). In particular, we extend UML use case and class diagrams by concepts of role-based access control (RBAC) and use graph transformation (GT) rules to model changes to data as well as the dynamic (re-)assignment of roles played by human actors. From these models we can derive specifications of the services required systematically in terms of pre- and post-conditions as well as communication scenarios modelling their interactions. We use the formal framework provided by GT to formalize consistency relations between the different parts of these models. The technique will be demonstrated with the use of a pharmacy scenario.**

***Index Terms*—human and software agents; service-oriented systems; visual models; graph transformation**

I. Introduction

In this paper, the problem being addressed is the ability to accurately model human actors as part of a service-oriented system. Like software agents, humans differ from standard software components by exhibiting properties such as reactivity, autonomy and pro-activity. The modelling of the orchestration between people and technical components should correctly display roles and responsibilities of all participants involved in the business process. These models of orchestrated systems need to take into account the non-deterministic and often non-predictable behaviour of humans. Unfortunately, methods and tools that are currently available within software engineering are not sufficient for addressing these issues.

Most formal approaches that have been developed so far in software engineering focus on technical components of a business process, by modelling how software and technical components will react to triggers issued by a software component and perform designated actions. However even if people are correctly instructed, they may or may not perform their allocated tasks.

Human actors can be categorized as a kind of autonomous agents with the capacity to regulate and coordinate their own behaviours [11]. In general, an *autonomous agent* is defined as a system situated within an environment that senses that environment and acts on it, over time, in pursuit of its own agenda, so as to effect what it senses in the future [7]. In other words agents are reactive to their environment and act in pursuit of their own goals. Autonomous agents can be broken down into different types of agents including software and human agents. Some of the main behavioural characteristics of autonomous agents are *reactivity, autonomy and pro-activity* [4]. *Reactivity* is defined as the capability to be sensitive to the environment and react to changes. *Autonomy* is defined as the ability to make one's own choices regarding the (non-)execution of a task, leading to non-deterministic [4]. *Pro-activity* is defined as the determination to reach a certain goal. These characteristics can positively or negatively impact a business process. For instance, because of autonomy, a person could refuse to perform an expected task for their assigned role, resulting in another person pro-actively taking over the role in order for a task to be successfully completed. Because of these shared characteristics, modelling techniques used for software agents could also be used for human actors.

We are proposing a modelling approach for human actors in service oriented systems (HASOS), based on a domain specific language (DSL) using concepts from *UML*, *role based access control (RBAC)* and *graph transformations (GT)*. These concepts are motivated by the fundamental behavioural characteristics of human agents and will be presented in detail in Section IV.

Modelling techniques for agent-based system often use UML notation because the mainstream object-oriented modeling technique provides a good basis for agent-specific extensions. Hence, our visual models are also based on UML, following Depke, et. al [4], [3] in extending it by a *rule-based approach* using *graph transformation* to specify agents' operations. The operational semantics of graph transformation allows simulation and analysis of dynamic reconfiguration, and graph transformation rules provide a good visual representations of non-deterministic structural changes in complex *business processes*. Graph transformation exhibit properties which reflect the characteristics of reactivity, autonomy and pro-activity. Reactivity is demonstrated through pattern matching in graph transformations, modelling how an actor can

978-0-7695-3943-0/09 $26.00 © 2009 IEEE
DOI 10.1109/SEEFM.2009.17

sense or observe its environment and react to it. Autonomy is demonstrated through the fact that a match is a requirement, but not an obligation to apply a rule. Pro-activity is shown by giving the person the choice to apply a variety of rules, among which a choice can be made. The model will also visually present a user's access rights to objects and roles by presenting rules for role assignment and reconfiguration of roles. Users who have the capability to assume a certain role, but not normally the right to do so, may have to overstep their permissions in exceptional cases. Potentially, such action will have to be justified later. It is therefore important to distinguish capabilities to perform an action or assume a rule, and the access rights to do so.

We first discuss related work in Section II. In Section III we will illustrate the use of HASOS on a pharmacy case study, defined using traditional UML techniques at business and architectural level. We will then introduce our new HASOS visual modelling language in Section IV and illustrate its use on the pharmacy scenario in Section V. Section VI will provide an analysis of the scenarios goals and dependencies. Section VII will present future work.

II. Related Work

We will present an analysis of the related work of BPMN, WS-HumanTask, BPEL4People and problem frames in comparison to HASOS specifications.

WS-HumanTask and BPEL4People are extensions of WSDL and BPEL motivated by requirements of specifying humans as part of service-oriented systems or processes. As the authors of BPEL4People stress, "the aspect of how people interact with business processes must be properly modeled" [8]. However, like their precursors, these are machine-readable XML languages format that lack a visual representation suitable for domain and business experts. Although BPEL4People captures human behaviours such as escalation and assignment of roles, extensions for ad-hoc processes have yet to be developed [5]. HASOS will also extend the current semantic capabilities of existing languages to address the challenges arising from the fact that human actors are harder to control than machines. This raises reliability and accountability issues.

On the other hand, the problem frames approach developed by Michael A. Jackson provides an abstract representation of the problem interaction for software requirements and specifications. It consists of three informal diagrams called: context diagrams, problem diagrams and problem frame diagrams. Context diagrams focus on customers' needs, responsibility and scope of authority by only displaying interfaces that are within the scope of the customer. HASOS also considers the viewpoint of the user by explicitly connecting the user to objects and roles that they have access rights to; however it also displays an overview of the surroundings by connecting the interfaces that go beyond their scope in a single model. The problem diagrams decompose the problem into subproblems and rights and privileges of membership can be exercised with the addition of annotations. Problem frame diagrams classify problems into three distinct domains: casual, biddable and lexical. The biddable domain consist of people and their "lack of positive predictable internal causality" [9], similar to the domain of human actors in HASOS. Both approaches emphasize the fact that humans may be ignorant of their operations and have unpredictable actions. The key difference is in the perspectives: HASOS provides a detailed look at the biddable domain whereas problem frame diagrams provide an overview of the interaction across three different domains. A person in this domain spontaneously causes events in a system without external stimulus, which is similar to our notion of pro-activity.

The formal approaches that have been developed so far in software engineering focus on technical components of a business process, by modelling how software and technical components will react to triggers issued by a software component and perform designated actions. However people cannot be guaranteed to do what they are told and may or may not perform their tasks, as demonstrated in the case study below.

III. Pharmacy Case Study

We will illustrate the use of HASOS on a pharmacy franchise. Shoppers Drug Mart Canada requires a systematic standardized method for all the pharmacies across Canada. All pharmacy owners go through an intense training on how to run the pharmacy. A visual model would add clarity to the training facilities and help visual learners to perceive the business process. This pharmacy scenario is a good example to test the language because it involves various actors in different roles using specialized software services such as security enabled barcode systems and printers connected to external sources.

We consider the case of an online refill prescription service. The refill prescription service involves the following steps: (1) customer transmits online prescription refill request; (2) the online service transmits payment to the customer's drug plan; if successful (3) a prescription label will be printed at the requested pharmacy location. If an exception occurs during the execution of the refill service, the customer and/or pharmacy will be notified for an alternative resolution. For instance if the customer's drug plan rejects payment then the refill transmission will be cancelled or another means of payment will be offered.

The *UML* activity diagram in Figure 1 illustrates the pharmacy business process. The swimlanes are used to partition the diagram based on the actors' roles. The actors involved in this problem domain are pharmacists, technicians, customers and doctors. Each actor is assigned a particular role and has various abilities. The doctor's role is to write a prescription or directly fax/phone in a prescription into a pharmacy. The customer's role is to bring the prescription from the doctor to the pharmacy or order a repeat prescription online or via telephone. The pharmacy technician's role is to receive prescriptions from customers, fill prescriptions, answer phone calls, and work the cash register. The pharmacist's role is to check filled prescriptions and counsel customers.

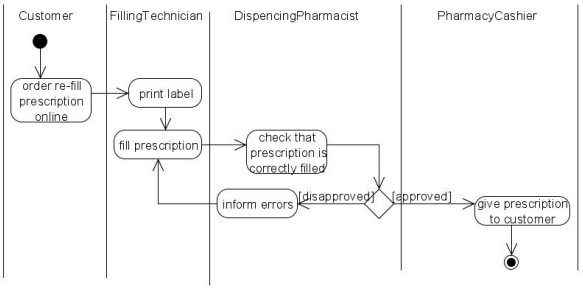

Fig. 1. Pharmacy Business Process in UML activity diagram

The pharmacy is divided into four stations: Entry Station, Filling Station, Checking Station, and Pick up Station. There is a minimum of one worker per station, and each worker is assigned unique bar-code identifiers. These bar-codes are used to track which worker completed particular tasks on particular prescriptions.

The human actors in this system have properties of the three characteristics of human agents of reactivity, autonomy and pro-activity. The pharmacy team can react to an increased customer demand. Each human actor involved has choices in the order they perform a task; for instance a technician can decide which prescription to fill first. This is an example of autonomy. A typical pharmacy business process is initiated with a prescription dispensing request. The prescription is entered into the pharmacy database and the corresponding bottle label is printed. The prescription is filled and checked. If any errors are found the prescription will be re-typed and/or re-filled. Once the medication is correctly filled, the customer is counselled and given the filled prescription. The pharmacy team are pro-active by working together to reach the goal of providing a correctly filled prescription to customers in a timely fashion.

There are various exceptional cases to this business process, which go beyond the scope of *UML* and standard business process models. For example, business process models do not address the assignment of roles to individual actors. Similarly, dynamic reassignment of roles is not presented either. For instance, in the pharmacy business process, this would be relevant in the exceptional situation where no pharmacy technicians are available. In this case, the pharmacist must take on all technician roles in addition to their pharmacist role. This exceptional case also reflects the three characteristic of human agents of reactivity, autonomy, and pro-activity behaviour. For instance, a pharmacist is proactive by taking over a role of technicians without being told to do so.

A UML sequence diagram was used to specify the communication between actors in a business process, as shown in Figure 2. The component diagram in Figure 3 was derived from the interactions between component interfaces in the sequence diagram. The service interfaces in (Figure 4) define the operations provided by each interface defined in the component diagram. These traditional UML diagram techniques depicted human agents as systems; however this is an inaccurate depiction because we cannot apply operations on humans, who will make an internal choice on how to react to external triggers.

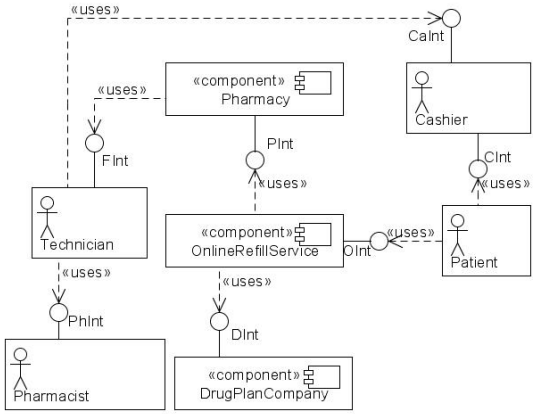

Fig. 3. Pharmacy architectural level in UML component diagram

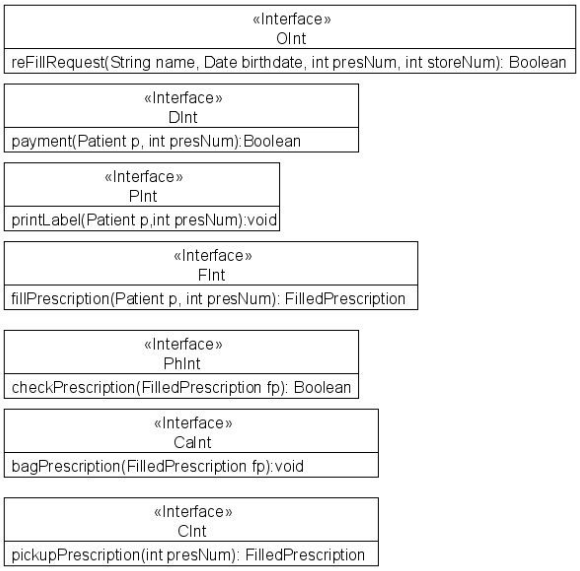

Fig. 4. Pharmacy architectural level in UML interfaces

The activity diagram, sequence diagram, component diagram and interfaces are all lacking requirements defined in the introduction. Role assignment is shown in the sequence diagram; however it does not include role reconfiguration. Requirements for roles are partially shown in the activity diagram; however there is no visual representation to show that the assignment has occurred. All the diagrams lack information on the user's role capabilities and permissions.

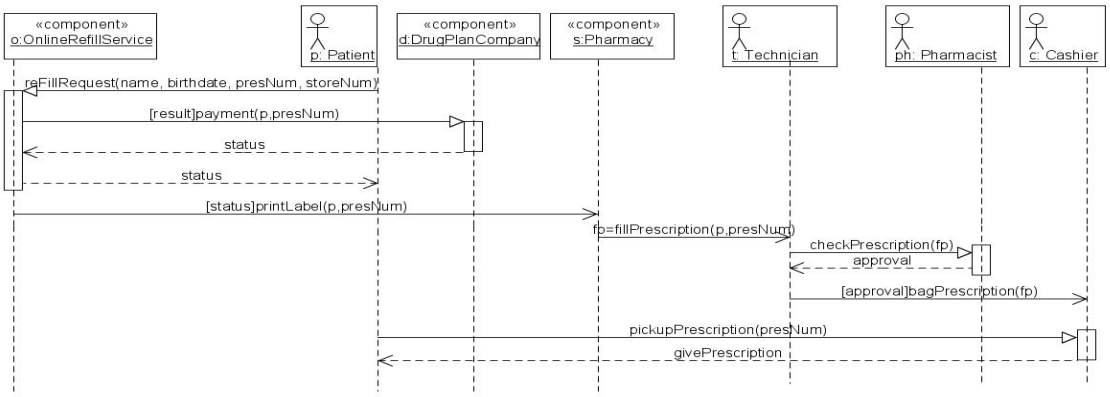

Fig. 2. Pharmacy architectural level in UML sequence diagram

IV. HASOS Language Framework

We defined the DSL using a (linguistic) metamodel for modelling the orchestration between people and technical components. The language syntax is illustrated using the pharmacy business domain defined in Section 13.

The HASOS metamodel is defined in Figure 5. For clarity the instance and type elements are located on opposite sides of the diagram.[1] Each element defined in the metamodel is an extension of either Class or Object type. Actor, Role and Case extend Class, whereas User, RoleInstance and CaseInstance are extensions of Object. The structure of the metamodel captures the concepts of role assignment, point of view of participants and access control. The relationship between User (instance level) and Actor (type level) is influenced by concepts of role-based access control (RBAC).

The influence of RBAC is due to the fact that the central notion in the meta model is the association of individuals to permissions via roles to permissions [12]. RBAC models provide a visual representation of the interactions between people and their corresponding task involvement in a business process. Roles represent competency, authority or responsibility to perform a specific task, which formulates the access control policy [12].

The $RBAC_3$ model in Figure 6 is composed of users, roles, permissions, sessions/objects, role hierarchy and constraint features. A user is defined as a human-being. A role is a job function or job title within the organization with additional semantics regarding authority and responsibility. Permission is defined as approval of a particular mode of access to one or more objects in the system. Objects are also known as sessions which represent the data within the computer system as both resource and data objects [12]. Users are not directly connected to permissions because this would weaken access control. Instead, the placement of roles between user and permission provides a greater control over access

[1]Since this is a linguistic metamodel the "instance-of" relationship appears within meta levels.

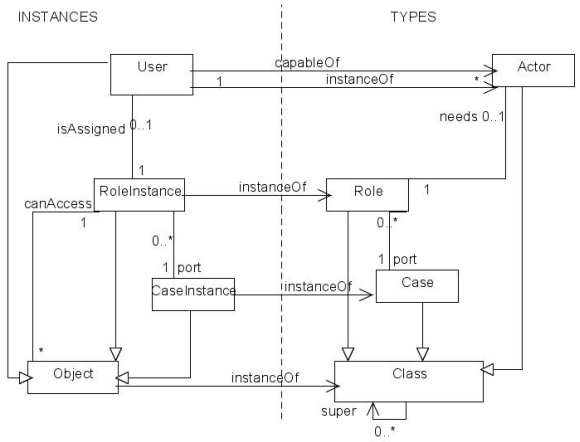

Fig. 5. Metamodel at M2 level

configuration. A subject (or session) is defined as a unit of access control, and a user may have multiple subjects active at the same time [12].

The modelling of the orchestration between people and technical components should correctly display roles and responsibilities of all participants involved in the business process; hence we decided to use $RBAC_3$ to help model the organizational structure of the business process.

Figure 7 is the abstract model at M1 level model, which is an "instance of" the M2 metamodel defined in Figure 5. This model is similar to a type graph in GT. It also formalizes Figure 8 and 9, which are the concrete model at M1 level.

The HASOS metamodel is composed of user, role and session elements for access control; however they are renamed to correspond with UML terminology. Role is represented as Actor and Session is as RoleInstance. Each Case type has a "port" connection to zero or many Roles. Each Role

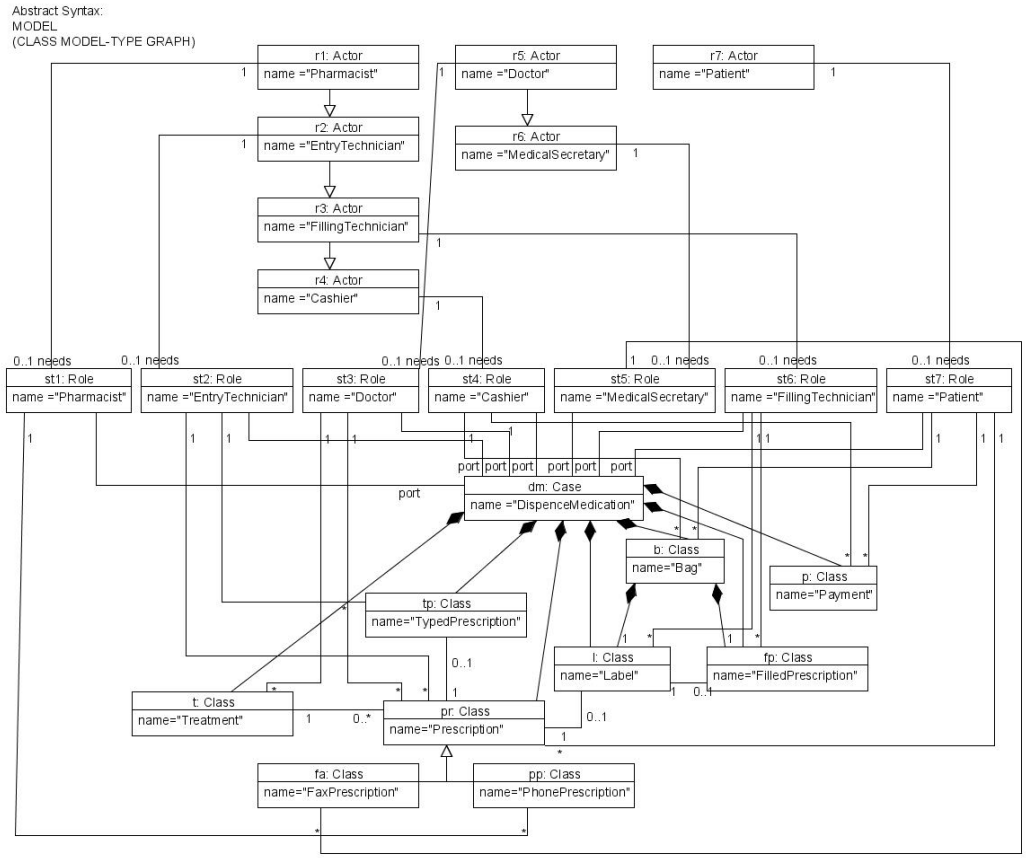

Fig. 7. Abstract Syntax at M1 level

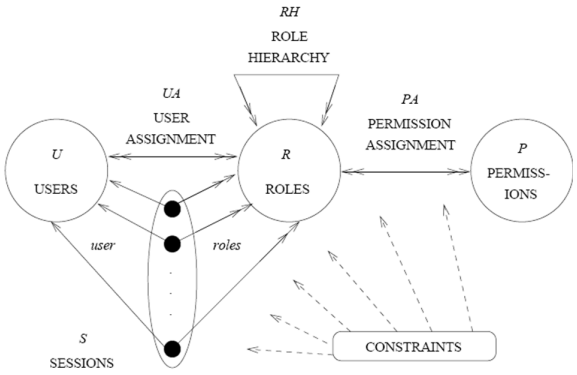

Fig. 6. $RBAC_3$ [12]

"needs" zero or one Actor. Object is an "instanceOf" Class, similarly User is an "instanceOf" Actor, CaseInstance is an "instanceOf" Case and RoleInstance is an "instanceOf" Role. The relationship between User and Actor has two associations between each other of "capableOf" and an "instanceOf", which represents a "memberOf" relationship.

These access rights influence the user-role assignment. An additional concept supported by the language displays the point of view of participants involved in a business process in relation to internal and external activities.

V. Modelling Notation

We expand UML use case to visualise the concepts laid down in the meta model. A use case diagram is composed of modelling elements of actors, use cases, association, generalization and the "includes" and the "extends" relationships. An *actor* is a role that a user, external to the system, plays in relation to the system [10]. The actor's description may be refined using *generalization*, as used in class diagrams.

UML diagrams are limited to depicting typical object-oriented systems.

> "Although the UML provides quite a considerable body of various diagrams that help to define an application, it is by no means a complete list of all the useful diagrams that you might want to use. In many places, different diagrams can be useful, and

you shouldn't hesitate to use a non-UML diagram if no UML diagram suits your purpose" [6].

Motivated by the development of use-case extensions for the SENSORIA Reference Modelling Language (SRML) [2], we employ use cases to depict Cases and Actor as Role elements. A use-case diagram describes "who the relevant users will be, the services they require of the system and the services they need to provide to the system" [10]. However they do not describe the internal workings of anything, hence the extension of its elements will provide additional visual detail to replace 'use case descriptions.' Our extension of use cases promotes the goal-driven characteristics of autonomous agents because the focus of use cases is on the goal.

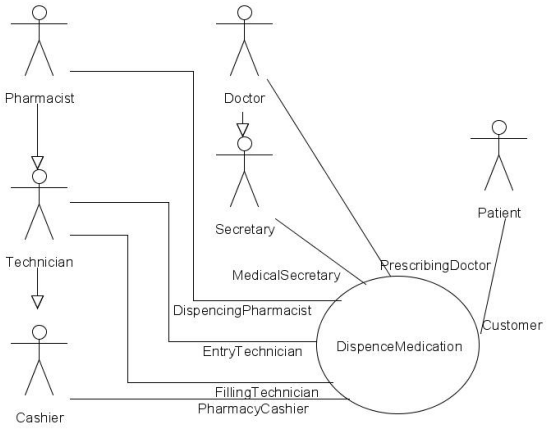

Fig. 8. Use Case Diagram

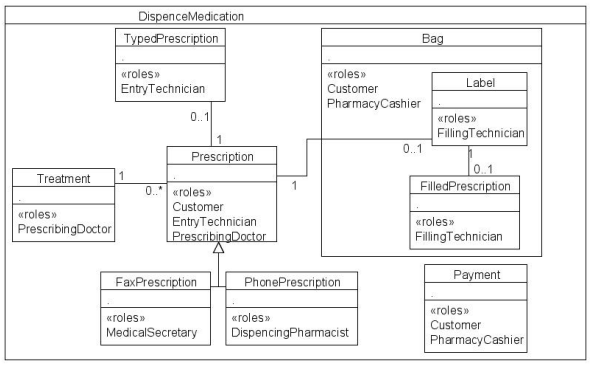

Fig. 9. Domain Model Class Diagram

The graphical notation is based on UML notation with minor extensions, as shown in Figure 8 and Figure 9. The use case diagram provides actor hierarchy and allowable roles for corresponding actors. In the use case diagram the roles are defined at the endpoints of the lines connected to the use case. The use case corresponds to a HASOS case which defines a detailed domain model. The keyword <<role>> is used within class elements to define access rights to elements within the class diagram. Additionally UML composition is visually represented with the containment of class/object elements.

The visual notation used to represent a generic user is a happy face. In order to show the specific relationship between user and actor a deviation of a happy face was used as shown in Figure 10. Illustrations of the use of this notation are shown in Figures 12 and 13.

Fig. 10. User graphical notation

Individual actions of our actors are described by graph transformation rules. Graphs occur at type and instance level, such as in UML class and object diagrams, respectively. Graph transformation is the rule-based modification on instance graphs. A graph transformation rule is composed of a left-hand side (L) and a right-hand side (R). The left-hand side represents the preconditions of the operation while the right-hand side represents the postcondition.

The rule in Figure 11 describes the "Fill Prescription" operation at the level of graphs representing meta model instances. The left-hand side of the rule requires a CaseInstance called "DispenceMedication" with composition of a "Label" Object. The CaseInstance must have one port to a "FillingTechnician" Role which "isAssigned" to a User that is an "instanceOf" the "Technician" Actor and "canAccess" the "Label" Object. With the negative condition stating that the "FilledPrescription" Object does not exist yet. The right-hand side defines the changes made in the transformation. The User who was an "instanceOf" a "FillingTechnician" generates a "FilledPrescription" Object and associates it with the "Label" Object.

Figure 12 (d) illustrates a more user-friendly presentation of the rule in Figure 11. Due to the use of graph transformation, the language can describe non-deterministic actions, reactive behaviour, as well as spontaneous actions, thus supporting the modelling of reactivity, pro-activity and autonomy of human behaviour. Tasks will be represented by a set of GT rules and the environment will be represented by instance graphs and type graphs. Reactivity will be addressed by allowing humans to select the left-hand side of a transformation rule that can be applied to the current state of the environment. Autonomy will be addressed by allowing the rules to be applied in any order. Pro-activity will be addressed by showing human decision to select rules in order to reach a chosen goal.

Note that a rule-based approach represents behaviour in terms of a set of "IF-THEN" rules. This form of modelling is in contrast to the *control flow* approach, which represents expected ordering of events. The control flow approach of BPEL4People models the expected order of tasks in a business process, whereas rule-based models support more directly a

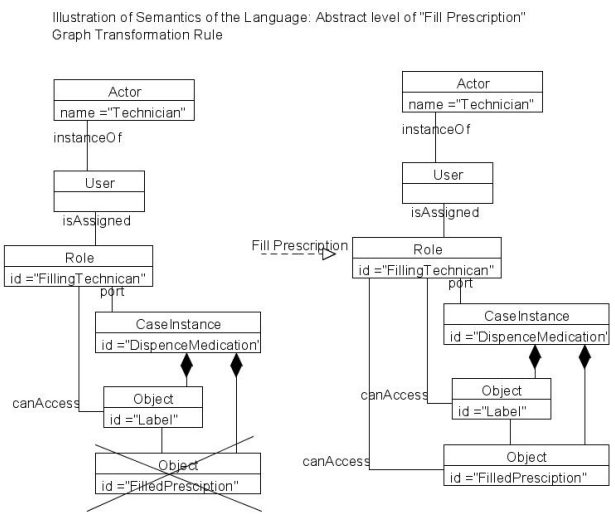

Fig. 11. Semantics illustration

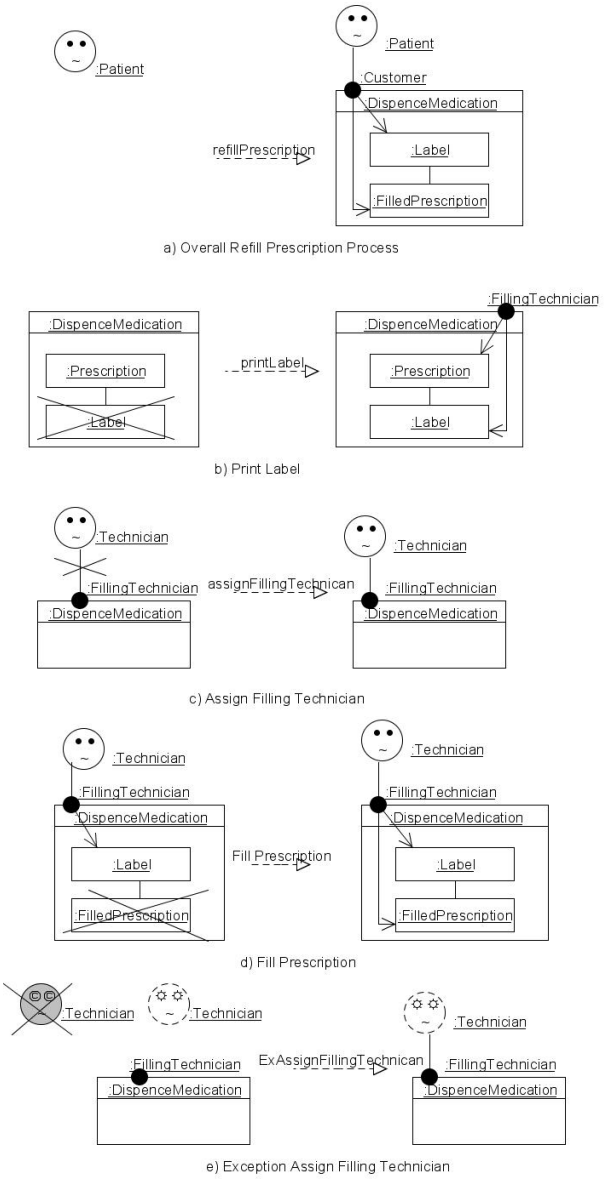

Fig. 12. Graph Transformation Rules

non-deterministic order of tasks in business process, intuitively allowing the human to select the rule to apply. The pharmacy scenario defined in Section III is illustrated in Figure 13, by applying various GT rules defined in Figure 12.

Note, that an exceptional rule was used in place of the standard on (c) in Figure 12 because no actor with "instanceOf" technician is available. In such a case, a person who was "capableOf" the role took it on instead.

VI. Analysis of Goals and Dependencies

Graph transformation provides a variety of analysis techniques, which make it a visual formal method rather than just a visual language. For example, consistency between the communication in the sequence diagram and the data transformation described by rules can be analyzed by determining dependencies between rules: These should reflect the order of messages represented in the corresponding sequence diagram defined in Figure 2. For example, both "Assign Filling Technician GT" and "Exception Assign Filling Technician GT" depend on "Print Label GT" because the transformation defines ":FillingTechnician" as a required role, which is one of the preconditions of both rules. Similarly rule (d) depends on rules (b), (c) and (e), because their post-conditions are subsets of its pre-condition. The order of these dependencies corresponds to the message order in the sequence diagram, therefore the model is consistent.

VII. Future Work

We intend to extend HASOS by adding standard BPMN notation such as timer, exception flow, ad hoc markers and then extend BPMN to include a visual representation of human involvement in business process specializing on fundamental notations for deadlines, priorities, and urgency (escalation). In order to expand the visual representation of human involvement in business processes we intend to illustrate the expressive nature of the XML-based languages of BPEL4People and WS-HumanTask. The current BPMN specifications does not capture the information in their specifications.

These specifications include timeouts, escalations and priorities. *Timeouts* are handled by triggering appropriate *escalation* actions [1]. HASOS will be expanded to include some of the elements located in WS-HumanTasks such as: state, priority, time stamp, and skipable indicators. If any deadline is not reached an escalation action can be triggered, which could invoke role reconfiguration.

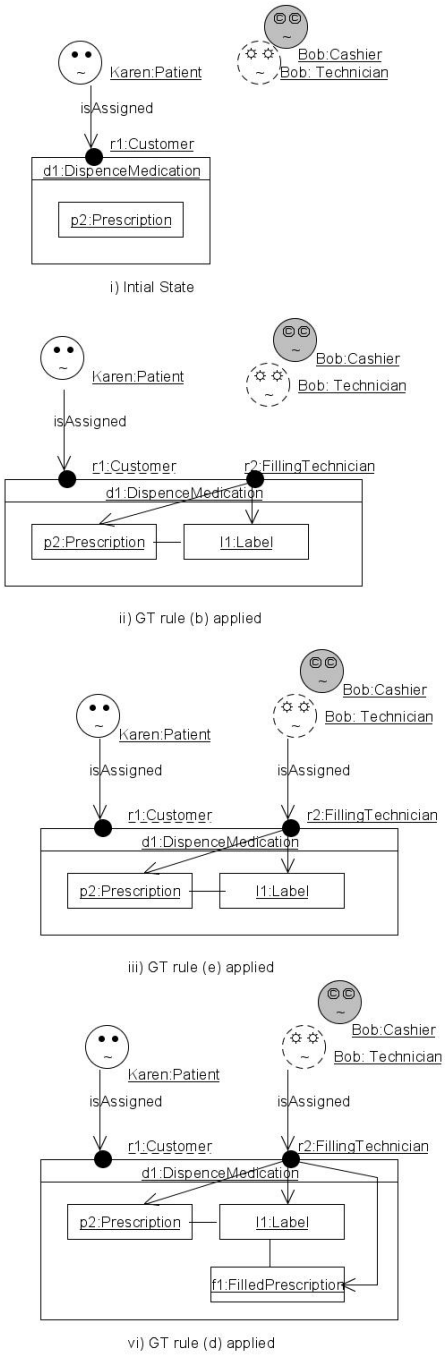

Fig. 13. Instance Graphs of Scenario in Section III

In order to create realistic depiction of role assignment, we intend to make a clear distinction between different types of achieved roles, by refining achieved roles into positions which reflect personal skills, abilities, and efforts. We can then research on the legality of temporally assigning particular achieved roles to people who are capable but not normally permitted to take on the role because they are at lower positions.

VIII. Conclusion

In this paper, we introduced a new visual modelling language HASOS which allows us to model humans as part of service-oriented system based on agent-based characteristics of reactivity, autonomy and pro-activity using a rule-based approach.

Acknowledgment

The authors would like to thank Georgina Donyina Pharmacist/Owner of store #729 Shoppers Drug Mart, Canada, for taking us through the pharmacy business process.

References

[1] Adobe, BEA, Oracle, Active Endpoints, IBM, and SAP. Web service human task (WS-HumanTask), version 1.0. Technical report, June 2007.

[2] Laura Bocchi, José Luiz Fiadeiro, and Antnia Lopes. A use-case driven approach to formal service-oriented modelling. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA*, volume 17 of *Communications in Computer and Information Science*, pages 155–169. Springer, 2008.

[3] Ralph Depke, Reiko Heckel, and Jochen Küster. Agent-oriented modeling with graph transformation. pages 105–119, 2001.

[4] Ralph Depke, Reiko Heckel, and Jochen Malte Küster. Formal agent-oriented modeling with uml and graph transformation. *Sci. Comput. Program.*, 44(2):229–252, 2002.

[5] José Fiadeiro and et al. Expanding service-oriented architecture to support human interaction (SOA4HU). September 2008.

[6] Martin Fowler. *UML Distilled Third Edition*. Addison-Wesley, Boston, Massachusetts, 2004.

[7] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, London, UK, 1997. Springer-Verlag.

[8] IBM and SAP. Ws-bpel extension for people bpel4people. Technical report, July 2005.

[9] Michael Jackson. *Problem frames: analyzing and structuring software development problems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[10] Tom Pender. *UML™ Bible*. Wiley Publishing, Inc., Indianapolis, IN, 2003.

[11] Jenny Preece. *Human-Computer Interaction*. Addison-Wesley, New York, 1994.

[12] Ravi S. Sandhu. Role-based access control, September 1997.