Modelling of Mobile Workflows with UML

Michael Decker
Institute AIFB, University of Karlsruhe (TH)
Karlsruhe, Germany
Email: decker@aifb.uni-karlsruhe.de

***Abstract*—Thanks to the advances on the field of mobile computing nowadays it is possible to realize workflow systems, which provide support for mobile activities, i.e., activities which are usually performed in situations where no stationary computer is available. It is obvious that the integration of mobile computers into workflow systems has a great potential especially for companies with a high portion of mobile workers. However, so far there is a lack of modelling techniques that allow to express mobile-specific aspects of workflows. In the following article we therefore introduce an extension to activity diagrams from the *Unified Modelling Language*, which allows to model different types of so called *location constraints*. Such a location constraint is a statement that defines where a particular activity of a workflow has to be performed or is not allowed to be performed. The proposed technique in particular supports *dynamic location constraints*, i.e., constraints that are defined during the runtime of a workflow instance. Further, different classes of anomalies that may occur in diagrams according to the proposed extension are also discussed.**

***Keywords*-Mobile Workflow Systems, Location-based Services, Business Processes, Activity Diagrams**

I. Introduction

A workflow is a set of partial-ordered activities that is performed with the support of a special computer system (so called Workflow Management System) to reach a particular business goal [2]. This goal could be the fulfillment of an order to perform some kind of on-site repair (e.g., fix machine in factory or in a private apartment). The individual activities required to reach this goal include *receive customer's call*, *dispatch service technician*, *perform on-site-repair* and *post-processing* (e.g., writing bill, evaluation). There is also a partial order defined for these activities that states which activities have to be finished before a particular activity can be started or if there are optional activities. Such a description of a workflow is also called a *workflow schema*. The individual invocations of that schema (e.g., order no. 123 by customer A., also termed a *case*) is a *workflow instance* of that schema. A workflow schema is a *mobile workflow* if there are instances that have activities that are typically performed using mobile devices like Personal Digital Assistants (PDA), smartphones or notebooks. The workflow sketched above as example is such a mobile workflow, because during on-site repair activities the concerned actors (mobile workers) do not have access to stationary desktop computers, so they have to use mobile computers to query technical data about components to be repaired, to order replacement parts or to write on-site reports about which workings had to be performed.

It is unquestionable that the employment of mobile devices for the enactment of workflows provides a great potential [3][4]: even while staying in the field mobile workers have access to latest information from backend systems (e.g., technical documentation, customer data, state of orders, availability of items). Further, it is possible to avoid media disruptions because mobile workers don't have to print every information they might need on-site and don't have to write reports and gathered data (e.g., orders for spare parts, measurement readings) onto paper forms that have to be entered into a computer system. This helps to avoid costs (for paper as well as working time needed for data gathering), errors (e.g., typos, double entries, lost forms) and the time delay between printing and information consumption or writing down the information and entering it into the backend information system is reduced.

The contribution of the article at hand is the introduction of an UML profile for activity diagrams that allows to cover important mobile-specific aspects, namely the constantly changing location of the mobile actors. To enable this so called *location constraints* are introduced: such a constraint is a statement attached to an activity that says at which locations the respective activity has to be performed or is *not* allowed to be performed. Not only static constraints are possible, i.e., constraints that are defined at the design time of a workflow schema, but also dynamic constraints, which are derived automatically during the runtime of a workflow instance. To actually enforce location constraints it is necessary that the workflow system knows about the mobile user's current location. For this several locating technologies are available: the most prominent one is the satellite-based *Global Positioning System (GPS)*, but there are many other technologies, see [5] or [6] for a good introduction and overview on this topic.

Location constraints for workflows are a variant of *Location-aware Access Control (LAAC)*. Access control is concerned with the decision if a user's request to perform a particular operation on a resource of an information system has to be granted or denied. Examples for pairs of operation and resource are "read on report.doc" or "write on database table1". When LAAC is employed the access decision is not only based on the user's identity and properties of the

resource but also on the current location of the user as determined by a locating technology. The novel feature of our extension for activity diagrams from the perspective of LAAC is that it supports process-aware location constraints, i.e., that it is possible to formulate location constraints based on the order of activities within a workflow. Further, it is even supported that activities of a workflow instance create location constraints for subsequent activities of the same workflow instance.

The work at hand is the extended version of a paper that was presented at the IARIA UbiComm-conference 2009 [1]. In addition to the original paper we also introduce some shortcut notations (Section V-B) and a notation for derived constraints (Section V-C). Further, we discuss the problem of anomalies with location constraints (Section VIII) and sketch how our concept of location constraints can also be applied for UML usecase diagrams (Section IX).

The following sections are organized as follows: In Section II the basics necessary for the understanding of the paper are covered. The underlying location model for our modelling approach is introduced in Section III. Based on these preparations the concept of *location constraints* is introduced in Section IV. The next Section V is devoted to the visual representation of the different types of location constraints. A detailed example of an activity diagram annotated with several location constraints can be found in Section VI. In Section VII it is elicited how UML's metamodel has to be extended to obtain a UML profile for location constraints. There are different types of anomalies that may occur if location constraints are used; this is the subject of Section VIII. Location constraints can also be used to annotate usecase diagrams which is explained in Section IX. A short survey on related work can be found in Section X before we conclude with a summary and outlook to future work in the final Section XI.

II. Preliminaries

In this section we describe the basics from the domain of *Location Aware Access Control* and the *Unified Modelling Language* that are required for the understanding of the discussion in this paper.

A. Location-Aware Access Control

When *Location-Aware Access Control (LAAC)* is applied the user's current position is considered by the component of the system that makes the access control decision [7]. For example, it could be enforced that a user is not allowed to access confidential data with his mobile computer when he stays at places that are deemed as unsafe, e.g., public places, countries where espionage has to be feared or regions where no secured wireless data transmission is available. So LAAC is a mean to tackle specific security challenges that come along with the employment of mobile computers. These challenges arise from the fact that due to their portability and size mobile computers often get lost or stolen. There is also the danger that someone looks over the shoulder of the legitimate user to gather data that is classified ("shoulder sniffing" or "shoulder surfing"). Further, wireless data transmission could get eavesdropped (passive attack) or even manipulated (active attack).

Another serious challenge in mobile computing are usability problems, since mobile devices only have a small display of limited quality and rudimentary means for data input (e.g., no full keyboard). Because of this the mobile user will appreciate it if data items and elements of the graphical user interface are hidden, when they are not relevant for him at his current location. For example, if a mobile service technician stays in a particular city all files concerning customers in other cities could be hidden.

B. Unified Modelling Language

The Unified Modelling Language (UML, [8]) is the result of the consolidation of several independently developed modelling languages (e.g., Object Modelling Technique (OMT) and Object Oriented Software Engineering (OOSE)) from the domain of software engineering. Nowadays UML is maintained by an industry consortium, namely the Object Modelling Group (OMG).

UML comprehends 13 different diagram types. On the uppermost level we can discern diagrams to describe *structural aspects* of software systems and *behavioural aspects*. A well-know example for the former type are *class diagrams* that are commonly used to describe data structures in object oriented programming languages (e.g., Figure 1). Activity diagrams belong to the latter type of diagrams. In the next section we show how these diagrams can be extended to enable expressing mobile-specific aspects.

The purpose of UML's activity diagrams is to describe workflows, i.e., to represent a set of activities and the relations between these activities. An individual activity is depicted as box with rounded edges that contains the name of the activity. The initial activity is a black circle and the final activity is represented as black circle within a bigger white circle. Arrows with solid lines connect these boxes to indicate which activities have to be finished before another one can be started; this way the control flow of the workflow is defined. To enable depicting conditional flows of activities there are control nodes (shown as diamonds) that are used to connect arrows. Further concepts in activity diagrams are swimlanes to group activities together (e.g., all activities that have to be performed by the same actor or organisation) or the forking/joining of the control flow for parallel activities.

Many other graphical languages for the modelling of workflows and business processes can be found in literature, e.g., Petri-nets, Event-driven Process Chains (EPC) or

Flowcharts. The distinguishing feature of UML activity diagrams for our purpose is that it provides an explicit defined meta-model and thus explicitly supports the definition of extensions.

III. Location Model

Before the actual extension of UML activity diagrams can be described we have to introduce a simple location model. For the sake of simplicity we consider only two dimensions, but it is simple to extend the model to support three dimensions which might be necessary to cover indoor scenarios where for example offices at different floors of a building have to be distinguished.

As geometric primitives there are *Point* and *Polygon* which both are extensions of the abstract superclass *AbstractGeometry* (see Figure 1). For polygons it is demanded that the lines of a given polygon don't cross each other. If a class is abstract this means that it is not possible to directly create instances of that class; to obtain an instance one has to create a concrete (i.e., non-abstract) subclass. A *circle area* is associated to one point that represents the center point of the circle; the radius of that circle is defined as member variable of class circular area and holds a double value that defines the radius of that circle in meters. *LocationInstances* are associated with exactly one polygon and belong to exactly one *LocationClass*. Location classes are used to group location instances that conceptually belong together, e.g., there might be location classes that represent cities, countries, districts or rooms within buildings. The concept of location classes and instances can also be found in the *Geographic Markup Language (GML)* in the form of feature types and features [9][10]. It is demanded that two polygons that belong to location instances of the same location class do not overlap spatially. Examples of location instances for location class *city* are *Malta* or *Berlin*. To exemplify this model also some instances of location instances and classes are drawn in the figure (surrounded by the box with the dotted line). It is *not* demanded that all the instances of one particular location class cover the whole reference space, i.e., that a given class provides an exhaustive classification. This is the case for class *city*, because not every point in a country can be assigned to a city, there are also rural areas. But a location class *country* could provide an exhaustive classification if the reference space is the whole area of a given continent; each point on that continent can be assigned to one instance of that class which represents a particular country.

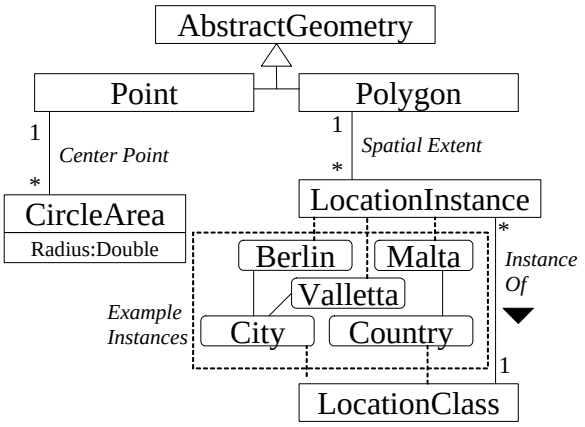

Figure 1. Location Model as UML-Class Model

IV. Location Constraints

In this section the concepts of *Location Constraints* and *Location Rules* are introduced.

A. Definition and Classification of Location Constraints

A *location constraint* is a statement about the location where one or more activities of a workflow schema or instance have to be performed or are not allowed to be performed. Constraints of the former type are called *positive constraints* while constraints of the latter type are *negative constraints*. These two types of constraints can be found in the classification of location constraints depicted in Figure 2.

A positive constraint could be demanded if a company wants to enforce that particular workflow activities dealing with confidential customer data are not performed outside the company's premises. Negative constraints will usually be employed if it is easier to express where something is not allowed than to enumerate the places where something is allowed; for example, if there are only a few countries in the world where certain software functions of a mobile workflow system should not be used (e.g., because of license or export restrictions) then a negative constraint will be defined that enumerates all these countries.

There is another dimension to classify location constraints which can also be found in Figure 2 and which is orthogonal to the distinction of positive and negative constraints:

Static constraints: These constraints are defined for the workflow schema *before* the runtime of the individual workflow instances. This implies that these constraints — which can also be called schema constraints — are enforced for all workflow instances that are created from that schema.

Dynamic constraints: They are defined *during the runtime* of a workflow instance are only valid for that instance. Further, we distinguish *external* and *internal* dynamic constraints: For external constraints the actual locations are not calculated by the mobile workflow system itself; rather, they are defined manually by a human operator during runtime (e.g., dispatcher, manager) or they are queried from an another information systems, e.g., a customer relationship management database that stores the addresses of all the customers of a company. When an internal constraint is defined then all the information needed by the mobile workflow system to calculate the actual location during the

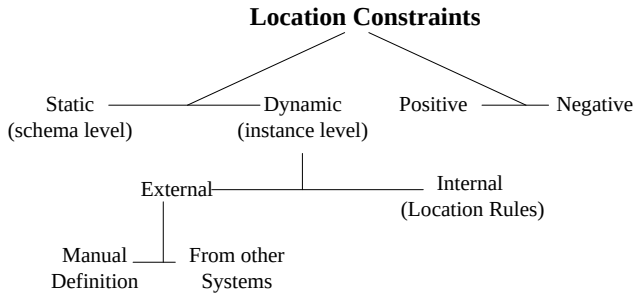

Figure 2. Different types of location constraints depicted as classification tree

runtime of the workflow instance has to be specified in the workflow graph. For this we propose so called *location rules* which are elicited in the subsequent subsection.

B. Location Rules

The basic concept behind so location rules is that based on the user's current location where he performs a particular activity called *trigger activity* a location constraint for another activity called *target activity* can be derivated. If the location constraint created by a rule is a positive one this means that both the trigger and the target activity have to be performed at the same location, so this is called *binding of (same) location*. If a negative constraint is created then the target activity cannot be performed at the same location as the trigger activity; this case can also be called *prohibition of same location* or *separation of locations*.

These two types of location rules were greatly inspired by the work of Bertino at al. [11]: they applied the well know security principles *Separation of Duties (SoD)* and *Binding of Duties (BoD)* [12] to the workflow domain. SoD for workflow management means that if a given user performed a particular activity of a workflow instance he is not allowed to also perform a particular other activity of the same workflow instance. The standard example for SoD is an approval workflow where the submitter of the proposal is not allowed to decide over his own proposal. An example for BoD is the policy *one face to the customer* that says that the actor who had the initial contact with a customer has to perform all other activities for the same workflow instance that involve communication with the customer. Bertino et al. even consider the case of inter-instance constraints, e.g., if Alice had to perform the activity *make decision* in a proposal workflow instance started by Bob then Bob isn't allowed to perform that activity in a subsequent proposal instance initiated by Alice to prevent collusion.

To define what is *the same location* the location model has to be employed. A location rule can refer to a location class; the instance of that class that contains the user's location when performing the trigger activity is the location that is used for the target location. It is also possible to define a radius that is used to calculate a circle around the user's location during the activation of the trigger activity.

Table I
EXAMPLE FOR AN IMPLICATION LIST WHICH MAPS LOCATION INSTANCES OF CLASS *country* TO LOCATION INSTANCES OF CLASS *city*

Trigger Location	Target Location
Germany	Munich
France	Munich
Malta	Valletta
...	...

Another approach is to have rules that create constraints pointing to a location different from the location where the trigger-activity was performed; for these rules with *implied locations* it is necessary to have lookup tables that map *trigger locations* to *target locations*. As example such an implication list is sketched in Table I: this table maps different countries to the city where the headquarter of an international company is. It is allowed that different countries are mapped to the same location, e.g., all the orders from either France or Germany have to be handled in Munich, because the number of orders from France is so small that it wouldn't make sense to operate a headquarter in France.

V. VISUAL REPRESENTATION

In this section first the basic elements of the visual representation are introduced. After this, shortcut notations are covered and the depiction of constraints derived based on location rules during the runtime of a workflow instance.

A. Basic Elements

As visual representation for the different types of location constraints and rules in UML activity diagram we propose the one which is sketched in the table that can be found in Figure 3. All the constraints in the left column are positive constraints or produce positive constraints, while the right column is devoted to negative constraints. The uppermost row shows static location constraints while all the other three rows show different kind of dynamic location constraints.

- All location constraints and rules are attached as dotted arrows to the boxes that represent the activities. On the dotted arrow there is a circle that holds a symbol which indicates the *mode* of the constraint: *Positive Static Constraint* and *Binding of same Location* are symbolized by "$=$", *Negative Static Constraint* and *Prohibition of same Location* are symbolized as "$\neq$", rules for *Implied Binding of Location* are symbolized as "$\Rightarrow$" and *Implied Prohibition of Location* are symbolized as "$\not\Rightarrow$".
- Static constraints are shown as arrows pointing from a parallelogram to the constrained activity. The parallelogram stands for a location instance and holds the name of that location.

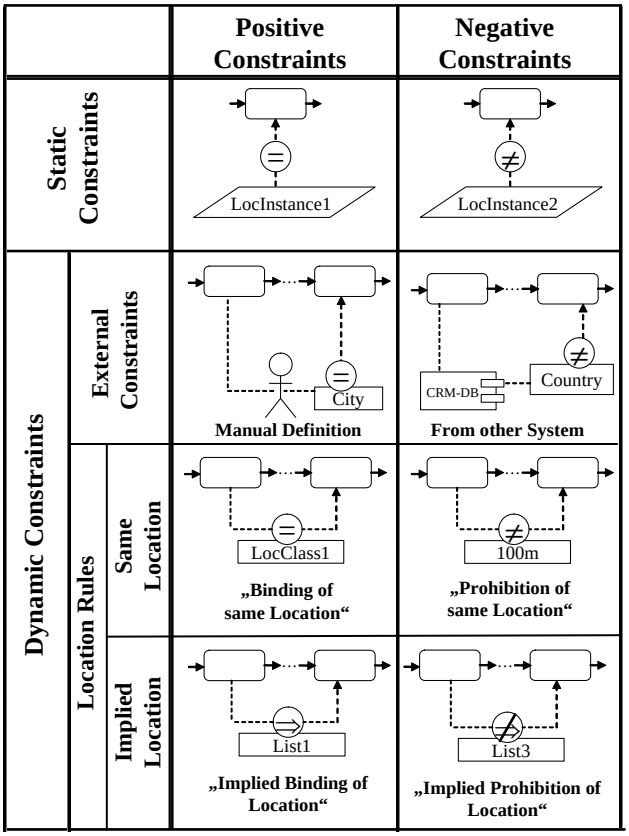

Figure 3. Different types of location constraints

- There are two types of external dynamic constraints, namely those where the actual constraint is entered manually by a human operator or is retrieved from another information system connected to the workflow system. Human operators are represented by a symbol that shows a little man. This symbol is also used to represent actors in UML usecase diagrams. An external information system as source for a dynamic location constraint is depicted by the symbol for a software component that was used in the deployment diagrams of UML 1.x. An external constraint is also depicited by a dotted arrow which connects two activities. The activity where that arrow starts is the activity which triggers the retrieval of the dynamic constraint. It is possible that the trigger activity is solely devoted to the retrieval of the location constraint; but the triggering could also be just a secondary effect of an activity. The target of the dotted arrow is the activity to which the retrieved location constraint will be assigned to. On the line of the arrow the symbol for the source of the constraint as well as a circle with hold the mode (positive or negative) and the granularity (location class or radius) is drawn.

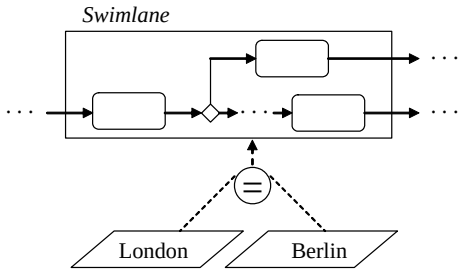

Figure 4. Location constraint for all activities in a swimlane

- A rule for the creation of dynamic constraints is shown as arrow pointing from the *trigger activity* to the *target activity* that will be bound to the derived location. For *same location rules* the granularity of what constitutes the *same location* is annotated by a box attached to the circle; the box can hold either a location class or a numeric value as radius; for *implied location rules* the box holds the name of the implication list. For example, if the location class *city* is used for the rule then the city that covers the user's current position is assigned as location constraint to the *target location*. If such a city cannot be found then no constraint is generated.

B. Shortcut Notations

We also devised some shortcuts which help to reduce the number of required elements to depict location constraints in some scenarios.

In Figure 4 several activities are contained within a swimlane (rectangular box). In UML swimlanes are usually employed to group activities together that are performed by the same actor. It is also possible to assign a location constraint to a swimlane, which then has to be enforced for all activities within that swimlane. This example shows also that it is possible to assign more than just one location with a static location constraint. If two or more locations are assigned as positive location constraints this means that the constrained activities have to be performed in one of these locations; in the example in Figure 4 this would mean that all the activities within the swimlane have to be performed either in London or Berlin.

When several locations are assigned to a negative location constraint this means that the activity can be performed everywhere but not at any of the assigned locations.

The fragment in Figure 5 represents the case where two different activities share the same location constraint. For these fragments the meaning is that both activities have to be performed somewhere in Canada; however, it would be possible to perform the two activity in different Canadian cities. Such a constraint could occur in practice if the pre- and post-processing of a workflow should be performed within a particular country, but all the other activities can be performed somewhere else in the world.

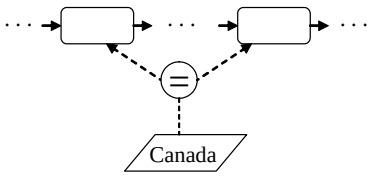

Figure 5. Single location constraint shared by two separate activities

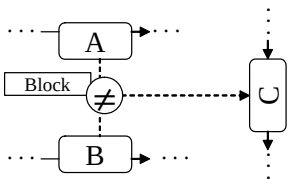

Figure 6. Location rule with two trigger activities

There are also shortcuts for location rules like shown in Figure 6: in this example one location rule has two trigger activities. The location instance of class *Block* (short for *Block of flats*) which contains the user's location when he performs activity A or B will define a negative constraint that forbids the execution of activity C at the same block. Analogous to this it is possible to have more than just one target activity for one location rule; this would mean that the location constraint created when that rule is triggered is attached to all the target activities at the same time.

C. Visualisation of derived constraints

While not necessary for the actual workflow modelling for the purpose of demonstration it is useful to be able to depict the static location constraint generated by a dynamic constraint; the generated constraint can also be called *derived constraint*

A static constraint generated by a dynamic constraint is depicted as a static constraint (as parallelogram) and attached with a dotted line to the box that indicates the granularity of the contraint. In Figure 7 this is shown for the case of a location rule, but the notation can also be used for external constraints. The rule in the figure creates a positive constraint that restricts the execution of the target activity to the city where the trigger activity was performed. In the depicted example the trigger activity was performed somewhere in Italy, so that an accordant static constraint was created.

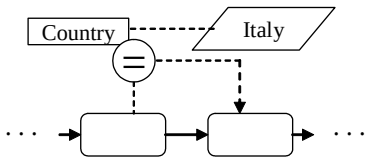

Figure 7. Depiction of derived constraint

VI. Example Scenario

To exemplify the application of our UML profile the workflow already sketched in the introductory section is elaborated in more detail and shown as diagram with location constraints in Figure 8. This workflow could be found in a company that provides technical maintenance services and employs many technicians which are sent to the customers' premises to perform on-site repair works of technical components.

An instance of the depicted workflow schema is created when a call center agent receives a telephone call from a customer. This activity has a static location constraint that points to all the location instances that represent the premises of call centers operated by the company. The incoming calls are routed to the call center that is the nearest with regard to the origin of the call, so the local service center that has to send a mobile worker to the customer's premises (dispatching) is defined based on a dynamic location constraint: the district in which the dispatching activity has to be performed is determined by an external application that makes this decision by evaluating the caller's phone number. The service center first sends an inspector to the site (e.g., factory, private residence, places with components of public infrastructure like pipes, electric transformation station, junction box) where the allegedly defective component can be found. If the inspector cannot find a technical defect the mobile part of the workflow is aborted and *Follow-up Office Work* is performed as final activity. However, if there is indeed a defect the inspector will order a mobile repair team to the location of the component using his PDA. The location where that team can perform the actual repair work is restricted with a dynamic constraint in form of a location rule (binding of same location) which says that the repair activities have to be performed not farther away than 150 meters from the point that was determined by the inspector's GPS device when he placed the order for the mobile repair team.

Since some components cannot be repaired on-site a possible branch of the workflow is *Shop Floor Repair* at a special repair shop operated by the company. To minimize transportation ways the location of this activity is constrained with another dynamic constraint that is based on an implication list; this implication list maps each region where the company provides its service to the nearest repair shop. If the activity in the shop floor is finished then another invocation of the activity *On-Site Work* has to follow since the repaired component has to be brought back and reinstalled. It is possible that the sequence of on-site work and shop-floor work is performed several times, e.g., if the defect is not solved after the first component that was repaired in the shop-floor is brought back.

The final activity is called *Follow-up Office Work* and deals with writing the bill, ordering new spare parts and

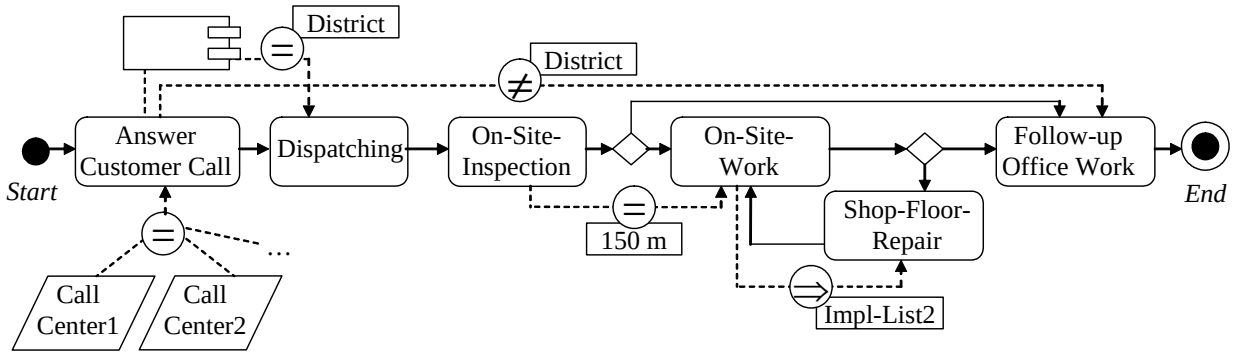

Figure 8. Example workflow with different kinds of location constraints

writing a report for the customer and the manufacturer of the component that had to be repaired. Since this activity also includes some kind of evaluation (e.g., how fast the customer's problem was solved, level of customer satisfaction) a dynamic constraint (prohibition of same location) is used to demand that the office center where this activity is performed is not in the same district as the one that initiated the workflow instance. The purpose of this rule is to ensure that employees knowing each other cannot collude to cover up mistakes that were made.

VII. UML Profile

We are now prepared to show how the proposed extension to UML activity diagrams fits into the meta-model of UML. For the sake of brevity only the most important new constructs are covered. In Figure 9 the package *MobileWorkflowProfile* is shown, that represents the new UML profile. Outside of the package some important classes from the UML meta-model are shown.

The profile also contains the location model which was introduced in Section III. Due to space restrictions only the parts of the location model are drawn that are necessary for understanding the relation of the location model and the rest of the model. There are also two new classes to model implication lists for rules with implied locations: *ImplList* and *ImplPair*. Each instance of *ImplPair* stands in association with two location instances to represent the *trigger location* and the *target location*. These instances of *ImplPair* (Implication Pair) represent the individual entries of an implication *ImplList* (Implication List).

The main connection between the UML meta-model and our profile is the extension relationship between class *Activity* and *ConstrainedActivity*. Instances of *ConstrainedActivity* have to be used if any kind of location constraint has to be assigned to an activity. For this each *ConstrainedActivity* stands in association with at least one instance of class *AConstraint*, which is the short form for *Abstract Constraint*, i.e., it is not possible to obtain direct location instances of that class. A flag named *isPositive* indicates if the *AConstraint* represents a positive or a negative constraint. There are two direct subclasses of *AConstraint*, namely *ADynConstraint* (short for *AbstractDynamicConstraint*, another abstract class) and *StaticConstraint*. Each instance of class *StaticConstraint* stands in association with at least one instance of *LocationInstance*. *ADynConstraint* as the second subclass of *AConstraint* has two direct subclasses: *ImplLocDynConstraint* and *SameLocDynConstraint*. *ImplLocDynConstraint* is associated with exactly one *ImplList*. Class *SameLocDynConstraint* stands in association with class *LocClass*. However, this association is an optional one because what is considered as "the same location" cannot only be defined based on a location class but also based on a radius. For the latter case there is also a member variable *radius* in class *SameLocDynConstraint* that is set to a value greater than 0 if the modeller thinks it is more appropriate to derive the "same location" using a radius than looking a location instance. For a given instance of *SameLocDynConstraint* it is not allowed to use both methods to define the "same location", i.e., either the radius is greater than 0 or there is an associated location instance. Using UML's *Object Constraint Language (OCL)* it is possible to express this formally [13]:

```
context SameLocDynConstr
inv     (not targetClass->isEmpty()) XOR
        (radius >= 0.0)
```

The keyword *context* is followed by the name of that class from whose perspective the following statement has to be viewed. For the given statement this means that *targetClass* has to be the name of an end of an assocation assigned to class *SameLocDynConstr* and that *radius* has to be the name of a member variable of that class. The keyword *inv* stands for *invariant*, i.e., the following boolean expression has to evaluate to *true* for all instances of the model. *IsEmpty()* is a function that returns *true* if for the considered instance there is no assocation to an object of *LocClass*.

Class *ActivityEdge* from the UML meta-model couldn't

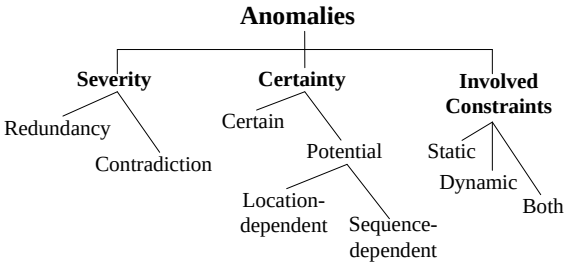

Figure 10. Different types on anomalies

be used to represent the arrows introduced in our profile to assign location constraints to activities because the edges in activity diagrams can carry tokens and we opted to have a different graphical representation (dotted arrows instead of solid arrows for the sake of better readability).

VIII. Anomalies of Location Constraints

When an activity diagram is annotated with location constraints according to the UML profile introduced in this paper it is possible that anomalies occur. In this section we will describe different types of such anomalies in terms of pairs of location constraints. The different dimensions considered to classify anomalies of location constraints are depicted in Figure 10.

The first dimension to classify the different types of anomalies is the *severity level*:

- A location constraint could make a **redundant statement**, i.e., the constraint could be removed without altering the behaviour of the mobile workflow system for any thinkable workflow instance. There are two subtypes of redundancy: the constrained activity is never reached or another constraint makes the same restriction or an even stronger one. The first case would occur if a static constraint is made to an activity that is never reached; this means that there is an inconsistency in the underlying workflow schema. Another case is a location rule whose target activity cannot be performed again once the trigger activity is reached; this means the location constraint generated by the trigger activity will never have any effect.
- Two location constraints could make **contradictory statements** for a given activity, i.e., there is no place where that activity could be performed by an actor according to both the constraints. The suggestive approach to deal with such contradictions would be to assign priorities to the individual constraints, so that in case of a conflict only the constraint with the higher priority is considered.

Redundant statements do not cause problems during the executing of a workflow instance but unnecessarily bloat the model. Further, the existence of a redundant statement might be a hint that there was an error during the elicitation of the required constraints.

Most workflows schemas have decision points, i.e., there are different sets of activities that are actually performed for a given workflow instance. For example, there might be a branch of a workflow schema that isn't executed for every instance. Further, the order of the activities can also differ between two workflow instances of the same workflow schema. Some anomalies may occur depending on the location of the actor while performing a particular activity. This leads to the next dimension for the classification of anomalies called *certainty*:

- A **potential anomaly** will not appear in every workflow instance of a given workflow schema. We distinguish two subcases of potential anomalies: the anomaly appears depending on the order and/or the set of actually performed activities (sequence-dependent) or merely based on the location where one or more activities where performed (location-dependent) no matter of the sequence of performed activities.
- A **certain anomaly** will appear in every workflow instance of a given workflow schema. So to detect this type of anomaly we just have to look at one potential workflow instance of that schema.

It also can be considered between what types of location constraints the anomalies arise:

- If two static constraints produce an anomaly this is called **intra-static anomaly**. For example, a positive static constraint assigned directly to an activity could demand that that activity is performed in London. However, this activity is contained in a swimlane which has also a static constraint which says that the activities are *not* allowed to be performed in England. Since London lies within England there is no place that satisfies both contradictory constraints.
- If both constraints involved into are dynamic constraints it is named **intra-dynamic anomaly**. As example we consider an activity that is the target of two positive location rules. During runtime the first rule creates a constraint that says that this activity has to be performed in Berlin; subsequently, the second rule creates a constraint that says that the same activity has to be performed in Spain. Since Berlin isn't a City in Spain this is a contradiction.
- The last case are **inter-static-dynamic anomalies** where both types of constraints are involved. So the pair of the conflicting constraints has one dynamic and one static constraint.

In Figure 11 three activity diagrams of tiny workflows are depicted which cover the three cases of inra-static, inter-static-dynamic and intra-dynamic anomalies:

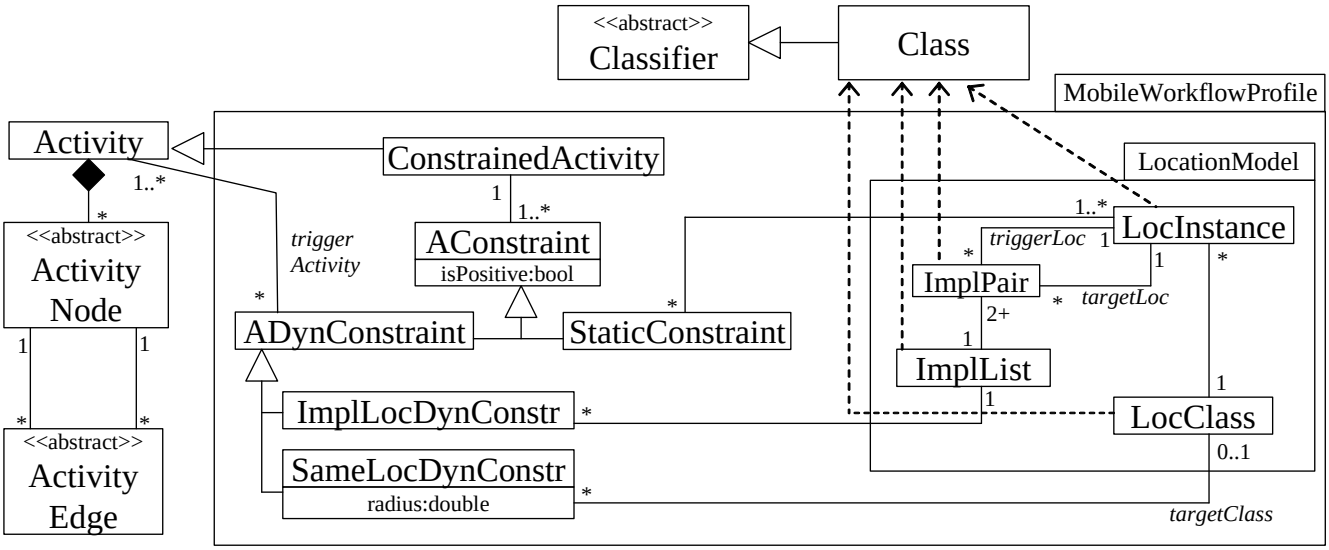

Figure 9. Extension of the meta-model of UML Activity Diagrams

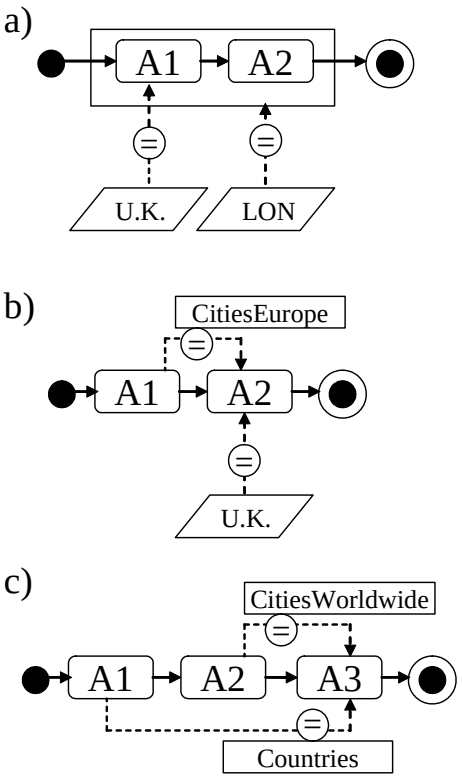

Figure 11. Examples for anomalies

- Diagram *a)* shows an intra-static anomaly. The swimlane that contains the two activities *A1* and *A2* has a constraint which confines the execution of these activities to the location *London*. Further, Activity *A1* has an individual constraint that restricts the execution of this activity to the location *United Kingdom (U.K.)*. This represents a redundancy, because *A1* is also confined by the constraint assigned to the swimlane, which is even stricter, since the location *London* is a subset of *U.K.* If the constraint assigned to activity *A1* would be omitted it wouldn't change semantics of the diagram. This case could occur if the constraint to *A1* was assigned long before the swimlane or the constraint for the swimlane was created and it was forgotten to remove the then redundant individual constraint for *A1*.
- Diagram *b)* shows an intra-dynamic anomaly: Activity *A2* is the target activity of a location rule and has also a static constraint. The location rule's trigger activity is *A1* and the derived location constraint will be an instance of the location class *Cities of Europe*. *A2's* location constraint points to the location *United Kingdom (U.K.)*, but not all European cities are located within the U.K., so this location rules represents are potential anomaly in the form of a contradiction.
- Diagram *c* shows an intra-dynamic anomaly: there are two location rules with the same target activity *A3*. The upper location rule (with trigger activity *A2*) assigns a location constraint that confines *A3* to a location instance of the class *Cities Worldwide*; the lower location rule with trigger activity *A1* assigns a location constraint that confines *A3* to a location instance of the class *Countries*. Since it is possible, that *A2* is

performed in a City that doesn't lie in the country where *A1* was performed this represents a potential anomaly in the form of a contraction. For example, it could happen, that *A3* has to be performed in *Germany* and *Lisbon* according to the derived location constraints, which is obviously impossible to satisfy, since Lisbon is not a German city.

In Figure 12 three more activity diagrams with potential anomalies are shown:

- In diagram *a)* the location constraint assigned to activity *A1* represents a redundancy (see also first diagram in Figure 11). However, the branch with the swimlane isn't executed in every run of this workflow this anomaly is a sequence-depend anomaly.
- Diagram *b)* shows a workflow schema with two concurrent branches. The potential anomaly is a contraction if *A2* is performed before *B2* in an European City outside the U.K. But if the execution of *B2* is started before *A2* is executed then the derived location constraint won't effect the execution of *B2* and so not contradiction occurs.
- The third diagram *c* in this figure has again two branches of which only one is executed. If the branch with activity *A* is executed in an European City this will lead to a contradiction, because the final activity *C* can only be executed in the *USA* according to the static constraint.

So all the anomalies in this figure are sequence-dependent. As example of a location-dependent anomaly we can consider diagram *b* in Figure 12: in this case a contradiction occurs if *A1* is executed outside the U.K.

IX. Location Constraints for Usecase Diagrams

Another diagram type which can be found in UML is for *usecase diagrams* [8]. The purpose of a usecase diagram is to show which functions (usecases) a software system should provide. Such a usecase is depicted as ellipse that containts a short textual description. It further shows different user roles and how these roles are connected to individual usecases. A role is depicted as human operator (the same symbol that was used for manual location constraints) and is connected by a line to each usecase he is allowed to perform. If the system to be developed consists of several subsystems (e.g., distributed system) then each subsystem is represented by a rectangular box that contains the usecases that are operated by this subsystem. Usecase diagrams have also the ability to show how individual usecases stand in relation to each other, e.g., if one usecase always invokes another usecase. However, this features isn't relevant for our consideration.

Usecase diagrams are usually employed at an early stage of the software development cycle. Based on the results of the usecase analysis activity diagrams can be developed.

Location constraints as defined in Section IV can also be used to annotate usecase diagrams. An example for such a diagram can be found in Figure 13: The left subsystem contains two usecases, namely *Create new order* and *Finalize order*; the right subsystem has also two usecases which are named *Create new account* and *Edit master data*. There are three roles in the diagram: *Travelling Salesman*, *Manager* and *Accountant*. It is possible to assign static location constraints to roles, usecases, system boundaries and the association between a role and a usecase. Further, usecases can also be the trigger or the target of a location rule: the *part of town* where the usecase *Create new order* is performed defines where the usecase *Finalize order* has to be performed. The rationale behind this rule is that a travelling salesman should only be allowed to finalize the order where he started it. Further, the role *travelling salesman* is restricted to Spain by a location constraint. This says that users of that role can only invoke usecases when they are within Spain. However, users with role *manager* are allowed to invoke the usecases they are assigned to without any spatial restriction. The third role *Accountant* has also a location constraint that prevents users with that role of invoking usecases when they are outside the rooms of the company's *accounting department*.

The subsystem at the right side of the figure has a static location constraint that confines all the usecases provided by that subsystem to the location *company premises*.

X. Related Work

In [14], an extension for UML activity diagrams is proposed that aims at modeling security requirements for business processes. The model introduces stereotypes to attach security requirements (e.g., privacy, access control, non-repudiation) to different elements in UML activity diagrams. For example, using this notation it is possible to assign a stereotype ≪privacy≫ to an activity partition (swimlane) to express that the disclosure of sensitive personal data has to be prevented during the activities contained by the within the swimlane.

Another profile for UML activity diagrams can be found in [15]. The purpose of this profile is to annotate workflows with concepts from the domain of business intelligence. Using this profile the modeller can express which activities require input from a data warehouse or data mart.

Baumeister et al. devised another extension of activity diagrams for modelling mobility [16]. But this approach is aimed at expressing how physical objects are moved by activities from one location to another and doesn't allow to formulate location constraints.

In literature some articles that propose location-aware access control models can be found. Notably most of them are extensions of Role-Based Access Control (RBAC, [17]) like *GEO-RBAC* [18], *LoT-RBAC* [19] or *S-RBAC* [20]. The most prominent aspects that distinguished these models is

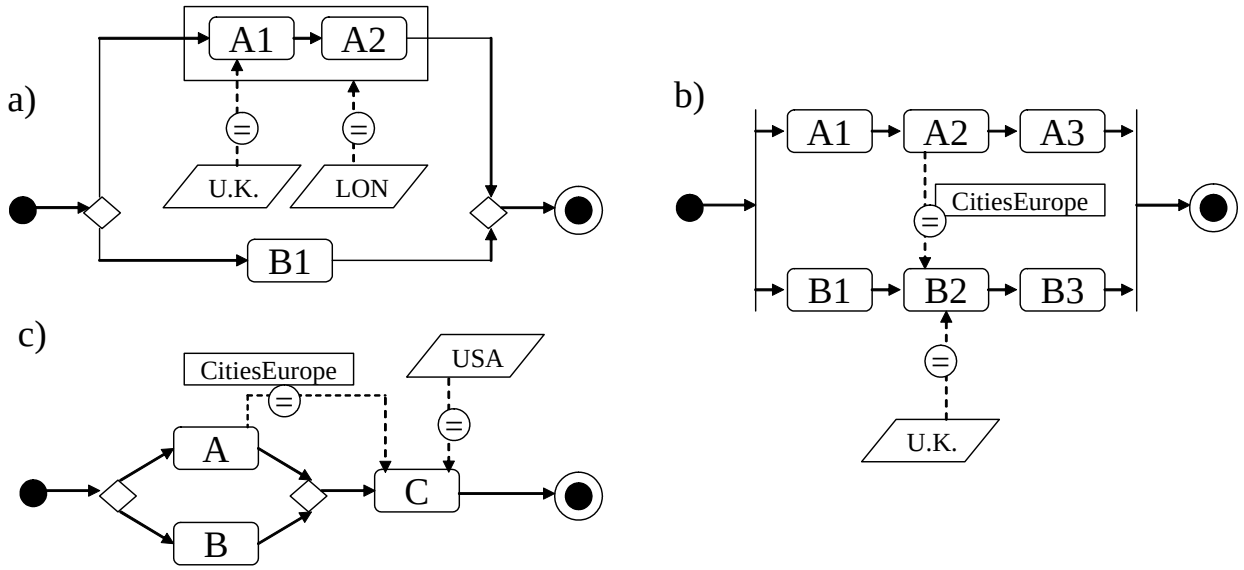

Figure 12. Examples for potential anomalies

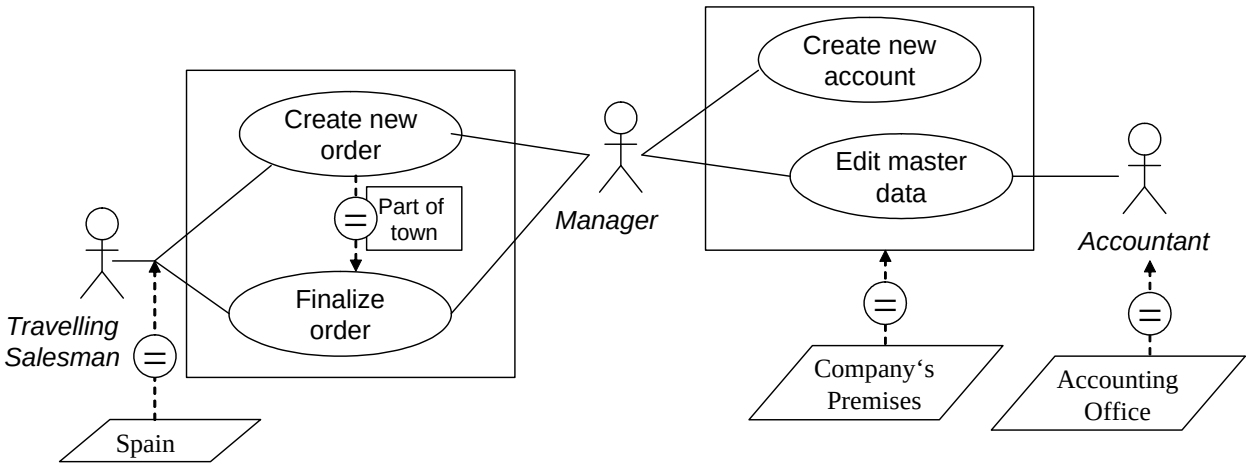

Figure 13. Usecase diagram with different types of location constraints

the subset of RBAC-components that can restricted with a location constraint. For example, in GEO-RBAC the location constraints are assigned to the roles itself, so depending on the mobile user's current location individual roles are switched on or off; in the S-RBAC-model location constraints can be assigned to the association between roles and permissions, so individual permissions of a role are switched on or off depending on the location. We surveyed these models in another article [7].

If access control decisions are based on the determination of a mobile user's location this leads to the question how trustworthy the employed locating system is. An attack with the intent to manipulate the location delivered by a locating system is called *location spoofing*. Such attacks can be performed by the possessor of the mobile computer (internal spoofing) or third party (external spoofing). In [21] we give an overview on different technical approaches for the prevention of location spoofing.

To the best of our knowledge we are aware of only one further paper by another group that deals with location constraints for workflows [22]. However, this model only allows to assign individual locations to activity nodes to state where

the respective activity is allowed to be performed. There are also location constraints assigned to the individual actors in the model. The focus of this work is the development of an algorithm to check if there are enough employees who can perform the individual activities of a workflow under the consideration of the respective location constraints.

In a recent paper we considered the problem of detecting inconsistencies in location aware access control policies [23]. However, this work focused solely on RBAC and didn't take any workflow-specific aspects into account. The basic idea is that it is possible to assign location constraints to several components of different type in a model at the same time. For example, a role *service technician* could have a location constraint so that this role can only be enabled in a certain region. Further, there could be also a location constraint assigned to a user that restricts his usage of the mobile information system to the city where he has to work. If the role *service technician* is assigned to that user it could occur that the intersection area of the two location constraints is empty, i.e., the user could never activate the newly acquired role. Such an "'empty assignment"' is a strong hint that the administrator of the system made a mistake and should be informed about this.

Further, the concept of *spatial coverage* as way to check the validity of location aware access control policies is introduced. For example, the coverage of a given role with respect to the entity *user* is the spatial extent of all the points where at least one user could activate that role according to the user's and the roles location constraints. If the coverage for the role *service technician* doesn't cover locations that should be served by the technicians then this again is a strong hint that something is wrong with the configuration of the access control model. In [23] we also describe several other types of spatial coverage for the purpose of model checking.

XI. Conclusion

In the article, an UML profile was introduced that extends activity diagrams to express spatial constraints — so called *location constraints* — concerning where individual activities have to be performed or are not allowed to be performed. Using our profile location constraints can be defined at the design time of a workflow schema, but it is also possible to define rules to automatically generate constraints during the runtime of a workflow instance.

Ideas for future work include to introduce further stereotypes to express mobile-specific constraints, e.g., to state minimum capabilities for the mobile computers to perform the activity, for example a minimum screen size or the availability of tamper-proof memory. Further, we envision the development of a graphical editor that supports drawing UML diagrams according to our profile; this editor should integrate functionalities to work with geographic data to define the spatial extents of location constraints.

Further, it would also be worthwhile to extend the location model with security labels in the sense of mandatory access control (MAC). To the best of our knowledge there are only two publication by other authors which deal with MAC-based location-aware access control [24][25]. However, these publications do not cover workflow-specific aspects. We also published an article on MAC-based location constraints for database management system [26], but again this work isn't workflow-specific. The basic idea of introducting security labels into our modelling approach would be to assign clearance levels to location, e.g., a particular country or building could have the clearance *Top Secret*, while other locations have only the clearance for the level *Secret*. Based on this classification location constraints could be assigned to a UML activity diagram. Such a constraint could state that a particular activity can only be performed when the mobile actors stays at a location that has a clearance of *Top Secret*, but not below; or a location rule could be used to assign a derived constraint so that the target activity's location has at a clearance level not lower than the location of the trigger activity.

In Section VIII several types of anomalies where discussed that can be found in activity diagrams when location constraints are applied. Since models should be free of such anomalies we are working towards an algorithmic approach to automatically detect such anomalies.

UML activity diagrams are by far not the only graphical language for the modelling of workflows. Another popular language for workflow modelling is the *Business Process Modelling Notation (BPMN)*, which is also maintained by the OMG [27]. We are also working on a BPMN profile for the expression of location constraints similar to the profile presented in this article [28].

Further, we are currently working on the application of the concept of location constraints for business processes from the domain of agriculture [29].

References

[1] M. Decker, "Modelling Location-Aware Access Control Constraints for Mobile Workflows with UML Activity Diagrams," in *The Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UbiComm 2009)*. Sliema, Malta: IEEE, October 2009, pp. 263–268.

[2] A. Oberweis, *Process-Aware Information Systems. Bridging People and Software Through Process Technology*. New York, USA, et al.: John Wiley & Sons, 2005, ch. Person-to-Application Processes: Workflow Management, pp. 21–36.

[3] M. Perry, K. O'Hara, A. Sellen, B. A. T. Brown, and R. H. R. Harper, "Dealing with mobility: understanding access anytime, anywhere," *ACM Transactions on Computer-Human Interaction*, vol. 8, no. 4, pp. 323–347, December 2001.

[4] M. Decker, "A Security Model for Mobile Processes," in *Proceedings of the International Conference on Mobile Business (ICMB 08)*. Barcelona, Spain: IEEE, July 2008.

[5] A. Küpper, *Location-based Services – Fundamentals and Operation*. Chichester, U.K.: John Wiley & Sons, 2007, reprint.

[6] J. Hightower and G. Borriello, "Location Systems for Ubiquitous Computing," *IEEE Computer*, vol. 34, no. 8, pp. 57–66, 2001.

[7] M. Decker, "Location-Aware Access Control: An Overview," in *Proceedings of Informatics 2009 — Special Session on Wireless Applications and Computing (WAC '09)*, Carvoeiro, Portugal, 2009, pp. 75–82.

[8] *Unified Modeling Language (OMG UML), Superstructure, V2.1.2*, Object Management Group, 2007.

[9] R. Lake, D. S. Burggraf, M. Trninic, and L. Rae, *GML. Geography Mark-Up Language. Foundation for the Geo-Web*. Chichester, U.K.: John Wiley & Sons, 2004.

[10] D. S. Burggraf, "Geography Markup Language," *Data Science Journal*, vol. 5, pp. 178–204, October 2006.

[11] E. Bertino, E. Ferrari, and V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 65–104, 1999.

[12] R. S. Sandhu, "Separation of duties in computerized information systems." in *Results of the IFIP WG 11.3 Workshop on Database Security (DBSec)*, Halifax, U.K., 1990, pp. 179–190.

[13] J. B. Warmer and A. Kleppe, *The object constraint language: getting your models ready for MDA*, 2nd ed., ser. The Addison-Wesley Object Technology Series. Boston, Massachusetts, USA: Addison-Wesley, 2003.

[14] A. Rodriguez, E. Fernández-Medina, and M. Piattini, "Towards a UML 2.0 Extension for the Modeling of Security Requirements in Business Processes," in *Third International Conference on Trust and Privacy in Digital Business (TrustBus)*, Krakow, Poland, 2006, pp. 51–61.

[15] V. Stefanov, B. List, and B. Korherr, "Extending UML 2 Activity Diagrams with Business Intelligence Objects," in *Proceedings of Data Warehosuing and Knowledge Discovery (DaWaK 2005)*, Copenhagen, Denmark, 2005, pp. 53–63.

[16] H. Baumeister, N. Koch, P. Kosiuczenko, and M. Wirsing, "Extending Activity Diagrams to Model Mobile Systems," in *Proceedings of NetObjectDays (NOD)*, Erfurt, Germany, 2002, pp. 278–293.

[17] D. F. Ferraiolo, R. Sandhu, E. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 224–274, 2001.

[18] M. L. Damiani, E. Bertino, and P. Perlasca, "Data Security in Location-Aware Applications: An Approach Based on RBAC," *International Journal of Information and Computer Security*, vol. 1, no. 1/2, pp. 5–38, 2007.

[19] S. M. Chandran and J. Joshi, "LoT-RBAC: A Location and Time-Based RBAC Model," in *Proceedings of the 6th International Conference on Web Information Systems Engineering (WISE '05)*. New York, USA: Springer, 2005, pp. 361–375.

[20] F. Hansen and V. Oleshchuk, "SRBAC: A Spatial Role-Based Access Control Model for Mobile Systems," in *Proceedings of the 7th Nordic Workshop on Secure IT Systems (NORDSEC)*. Gjovik, Norway: NTNU, 2003, pp. 129–141.

[21] M. Decker, "Prevention of Location-Spoofing. A Survey on Different Methods to Prevent the Manipulation of Locating-Technologies," in *Proceedings of the International Conference on e-Business (ICE-B)*. Milan, Italy: INSTICC, 2009, pp. 109–114.

[22] R. Hewett and P. Kijsanayothin, "Location contexts in role-based security policy enforcement," in *Proceedings of the 2009 International Conference on Security and Management (SAM'09)*, Las Vegas, Nevada, USA, 2009, pp. 404–410.

[23] M. Decker, "An Access-Control Model for Mobile Computing with Spatial Constraints - Location-aware Role-based Access Control with a Method for Consistency Checks," in *Proceedings of the International Conference on e-Business (ICE-B 2008)*. Porto, Portugal: INSTICC, July 2008, pp. 185–190.

[24] U. Leonhardt and J. Magee, "Security Considerations for a Distributed Location Service," *Journal of Networks and Systems*, vol. 6, no. 1, pp. 51–70, 1998.

[25] I. Ray and M. Kumar, "Towards a Location-based Mandatory Access Control Model," *Computers & Security*, vol. 25, no. 1, pp. 36–44, 2006.

[26] M. Decker, "Mandatory and Location-Aware Access Control for Relational Databases," in *Proceedings of the International Conference on Communication Infrastructure, Systems and Applications in Europe (EuropeComm 2009)*, ser. LNICST, R. M. et al., Ed., no. 16. London, U.K.: Springer, August 2009, pp. 217–228.

[27] OMG, *Business Process Model and Notation (BPMN) v. 1.2*, Object Management Group, January 2007.

[28] M. Decker, H. Che, A. Oberweis, P. Stürzel, and M. Vogel, "Modeling Mobile Workflows with BPMN," in *Proceedings of the Ninth International Conference on Mobile Business (ICMB 2010)/Ninth Global Mobility Roundtable (GMR 2010)*. Athens, Greece: IEEE, 2010, pp. 272–279.

[29] M. Decker, D. Eichhorn, E. Georgiew, A. Oberweis, J. Plaßmann, T. Steckel, and P. Stürzel, "Modelling and enforcement of location constraints in an agricultural application scenario," in *Proceedings of the Conference on Wireless Applications and Computing 2010 (WAC 2010)*, Freiburg, Germany, 2010, accepted, to appear.