

GEOSPATIAL CONTEXT AWARENESS IN BUSINESS PROCESS MODELING

by

MANUEL CORREA

(Under the Direction of Krzysztof J. Kochut)

ABSTRACT

The purpose of this thesis is to describe the geospatial context in a business process model. The study presents a proposal to add and to validate geospatial constraints in business processes. Another goal was to define geospatial reasoning through business rules and to describe the external geospatial context in a business process. The research proposes a new concept to build business process with location awareness through a prototype system using as example the Emergency Response systems.

INDEX WORDS: Business Process Model Notation, Business Rules, Business process context awareness, Geospatial modeling, Geographic Information System, Workflow Systems, Business intelligence.

GEOSPATIAL CONTEXT AWARENESS IN BUSINESS PROCESS MODELING

by

MANUEL CORREA

B.S., Telematics Engineer, Universidad Icesi, Colombia, 2006

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial

Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2012

© 2012

MANUEL CORREA

All Rights Reserved

GEOSPATIAL CONTEXT AWARENESS IN BUSINESS PROCESS MODELING

by

MANUEL CORREA

Major Professor: Krzysztof J. Kochut

Committee: John Miller
Prashant Doshi

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2012

DEDICATION

To my wife, family, friends, and professors thank you.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vii
LIST OF FIGURES	viii
CHAPTER	
1 INTRODUCTION.....	1
2 BACKGROUND.....	3
2.1 Workflow Systems	3
2.2 Business Process Modeling Notation (BPMN 2.0)	6
2.3 Business Rules	12
2.4 Context Awareness in Business.....	15
2.5 Geospatial Modeling.....	17
3 GEOSPATIAL AWARENESS IN BUSINESS PROCESSES.....	22
3.1 Problem description	22
3.2 Motivating scenario	23
4 RELATED WORK	26
5 GEOSPATIAL MODELING IN BUSINESS PROCESSES.....	30
5.1 GCL: Geospatial Constraint Language for BPMN.....	30
5.2 GDL: Geospatial Domain Language for Geospatial Business Rules .	43
5.3 Geospatial Context for Business Processes	51

6	GEOFLOW: SYSTEM ARCHITECTURE	54
6.1	Geospatial Engine layer	55
6.2	Business Process and Business Rules Layer	56
6.3	Context Awareness Layer.....	58
7	GEOFLOW: PROTOTYPE IMPLEMENTATION AND EVALUATION.....	61
7.1	Implementation details	61
7.2	Emergency Response System Use Case and Evaluation.....	67
8	CONCLUSIONS AND FUTURE WORK	81
8.1	Conclusions	81
8.2	Future work.....	83
	REFERENCES	85
	APPENDICES	
A	GCL USE CASES	88
B	GDL USE CASES	98

LIST OF TABLES

	Page
Table 1: BPMN 2.0 Event types [9].....	9
Table 2: Basic Geospatial Operations by Category	19
Table 3: GCL constraints extension for BPMN.....	39
Table 4: Business rules in an Emergency system workflow.....	50

LIST OF FIGURES

	Page
Figure 1: Basic Workflow control patterns.....	4
Figure 2: BPMN 2.0 Tasks	7
Figure 3: Tasks Markers [9].....	7
Figure 4: Sub-process [9].....	8
Figure 5: BPMN 2.0 gateways [9]	10
Figure 6: Single data object, Collections, Stores. Data Input/Output [9]	10
Figure 7: Connecting data to tasks. Connecting data to connecting elements [9]	11
Figure 8: Sequence. Messages. Association	11
Figure 9: Pool with two lanes [9].....	11
Figure 10: Group Artifact. Annotation Artifact [9]	12
Figure 11: Rete graph of assessment situation rule.	14
Figure 12: Context model	16
Figure 13: Point, Line, and Polygon	18
Figure 14: Emergency Management Business Process	25
Figure 15: Geo-Marker. BPMN with two tasks marked with geospatial data.....	37
Figure 16: Graphical representation of BPMN with Geo markers	38
Figure 17: Geospatial Context Awareness Business process overview.....	54
Figure 18: Geospatial Engine layer.....	55
Figure 19: Business Rules Engine with Geospatial reasoning support.....	56
Figure 20: GDL to Production Rule.....	57

Figure 21: Business Process Engine with Geospatial constraints support.....	58
Figure 22: Context awareness layer	60
Figure 23: Overview of the prototype system.....	61
Figure 24: UML classes diagram for Geospatial Engine.....	62
Figure 25: Process instance and GCL Engine interaction	64
Figure 26: Web Application used to interact with the GEOFlow system.....	67
Figure 27: Emergency situation representation in the map	69
Figure 28: Emergency response system in BPMN with GCL constraints.....	70
Figure 29: Output of Tornado Assessment and School Notification task in the Emergency response system BPMN	71
Figure 30: Output of Hospital preparedness task and the Activate shelter task in the Emergency response system BPMN	72
Figure 31: Output of Alert community task in the Emergency response system BPMN .	73
Figure 32: Tornado in the south of Atlanta.....	74
Figure 33: First set of rules for tornado emergency.....	75
Figure 34: Complex rule for tornado emergency. Assigns hospital to each school based on proximity.....	75
Figure 35: Output after firing the Emergency response rules.....	76
Figure 36: Business Process with Geospatial context awareness	78
Figure 37: GeoEvents register to the GEOFlow system through the Map application	78
Figure 38: Business rules used to filter GeoEvents	79
Figure 39: Business Process with context awareness output	80
Figure 40: GEOFlow RSS feed for business process events	80

Figure 41: Proximity relationships in BPMN	89
Figure 42: Map representing the proximity constraint in the BPMN process	90
Figure 43: Output of the execution of proximity relationships in BPMN	90
Figure 44: Geometric task's relations in BPMN.....	91
Figure 45: Map representing the geometric constraint in the BPMN process	91
Figure 46: Output of the execution of geometric relationships in BPMN	92
Figure 47: Measurement constraints for BPMN	93
Figure 48: Map representing the measurements constraint in the BPMN process	93
Figure 49: Output of the execution of measurements relationships in BPMN	94
Figure 50: Geographic network relationships in BPMN	95
Figure 51: Map representing the network constraint in the BPMN process	96
Figure 52: Output of the execution of geographic network relationships in BPMN	96
Figure 53: Filter constraint in BPMN	97
Figure 54: Output of the execution of the filter relationships in BPMN	97
Figure 55: Map representing the filter constraint in the BPMN process	97
Figure 56: Proximity rules in GDL.....	98
Figure 57: GeoFacts for proximity rules.....	99
Figure 58: Output after firing proximity rules	99
Figure 59: Geo Facts for geometric relationships rules	100
Figure 60: Geometric relationships rules in GDL.....	100
Figure 61: Output after firing geometric relationships rules	101
Figure 62: Geographic point network relationships rules in GDL	102
Figure 63: Geo Facts Geographic point network for rules	102

Figure 64: Geo Facts Geographic road network for rules.....	103
Figure 65: More Geo Facts Geographic road network for rules	103
Figure 66: Geographic road network relationships rules in GDL.....	103
Figure 67: Output after firing Geographic network (North of, South of, East of, West of) relationships rules	104
Figure 68: Output after firing road network relationships rules	104
Figure 69: Spatial grouping relationships rules in GDL.....	105
Figure 70: Geo Facts and polygon constant spatial grouping for rules	106
Figure 71: Output after firing spatial grouping relationships rules.....	106

CHAPTER 1

INTRODUCTION

One of the main goals of Service Oriented Architecture (SOA) is offering the ability to compose services in order to create complex business process representations. Business Process Modeling has gained popularity in the past several years. This enthusiasm has led to the development of more complete standards to represent the core of a business and its internal and external interactions with the environment. BPMN 2.0 is a good example of one of the recent efforts of the Object Management Group (OMG) to provide a full standard, which represents business processes in a graphical way as well as its semantics.

In the recent years, Geospatial technologies have gained a lot of popularity and usage in the Web. Today, the Web is the place of integration for Geographic Information Systems (GIS). Geospatial analysis is an important factor to make effective decisions within a business process. Location awareness is a key component in many devices and software frameworks. The necessity of businesses to embrace geographic data is evident and the main players in the GIS market are contributing actively with tools and APIs aiming to integrate GIS data into enterprise systems. This process has enabled new concepts of business intelligence, location awareness, and geospatial analysis which are now part of the decision making process for any business. This research addresses the main issues of adding geospatial modeling in the three main components of a business

process: business process model, business intelligence, and context of the business process.

This thesis starts with an overview of workflows theory and workflow patterns, an overview of BPMN 2.0, an overview of Business rules system, an overview of context awareness in business process, and an overview of geospatial modeling in Chapter 2. Chapter 3 describes the problem and presents a motivating example. Chapter 4 gives an overview of related work. Chapter 5 introduces the concepts of a business process and geospatial constraints, geospatial reasoning through business rules, and geospatial context awareness. Chapter 6 presents the architecture of a prototype system named GEOFlow. Chapter 7 shows the details of the implementation through different use cases and a real life example of an emergency response system. Chapter 8 concludes the work and opens the discussion for future work.

CHAPTER 2

BACKGROUND

2.1 Workflow Systems

According to the Workflow Management Coalition, a workflow is concerned with the automation of passing documents, information, and tasks between participants in order to achieve a business goal [1]. A workflow system is a series of tasks which join together to represent a real life scenario in a business process. A workflow fully describes the essence of a business and all its participants. Workflow systems have evolved from the definition of processes with Petri-net [2] theory to the development of Web Service composition and business models. One of the main contributions in the workflow community has been the definition of workflow patterns which delineate the fundamental requirements of workflow systems. This workflow definition is divided into control flow [3], data flow [4], resources [5], and exception patterns [6]. The control flow patterns describe the way that tasks are organized together. These patterns were designed through studies of different workflow processes and modeled in business processes. The control flow patterns are divided into six main categories: basic, advanced branching and synchronization, structural, multiple instances, state-based, and cancelation patterns. The basic control patterns describe the basic constructs in a workflow which are sequence: two or more tasks joining together in sequence; parallel split: an AND split, which entails the execution of two tasks in parallel; synchronization: joins two parallel tasks into

another task; exclusive choice: a point of choice to execute either of two tasks also known as XOR; simple merge: merges two tasks which were split by an XOR operation.

Figure 1 shows the five basic patterns in BPMN notation.

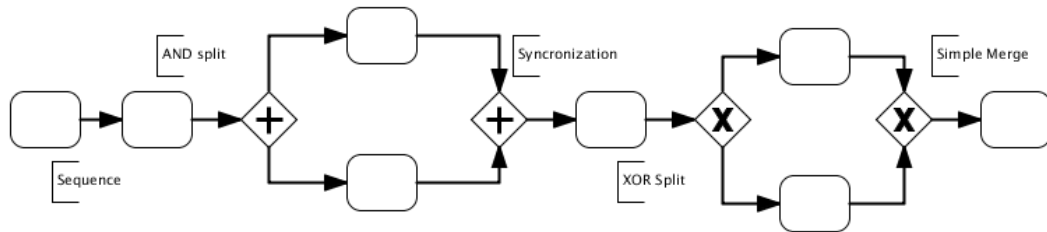


Figure 1: Basic Workflow control patterns

Other important patterns in the next categories are the multi-choice patterns, which are part of the advanced branching patterns; arbitrary cycle, which are part of the structural patterns; deferred choice, which are included in the state-based patterns that describe the concept of events in workflows; cancel activity, which is a part of the cancellation patterns. The theory of control workflow patterns describes the basics for modern structural workflows, and these patterns are highly regarded in formal standards.

Data patterns describe the data flow in workflow systems. These patterns are divided into four groups, which are: data visibility, data interaction, data transfer, and data-based routing. Data visibility relates to the way that elements of the workflow data see the data, data interaction concentrates on the manner which the data is communicated between elements of the workflow, data transfer considers the means of transferring data from one activity to another, and data-based routing deals with how the data affects the normal flow of the process. Within the data visibility group, the most relevant patterns are the task data pattern which describes the data that is input into a specific task, the

scope data pattern which describes the visibility of the data according to the tasks in the workflow, the workflow data pattern which describes the global data used in the process, and the environment data patterns which describe the external data that affects the process. The relevant patterns in the data interaction group are task to task patterns, multiple instances patterns, task to environment patterns, and environment to task patterns. In the data transfer group, the most relevant patterns are data by value in incoming and outgoing patterns, copy in and copy out patterns, and data transformation patterns. The data-based routing group includes precondition and post-condition data patterns, event based task trigger patterns, and data-based routing patterns. The resource patterns illustrate how the workflow system is implemented. These patterns describe in detail the implications in execution time and resource allocation for a workflow process. Finally, exception patterns describe how to handle exceptions in workflow systems. These patterns discuss issues such as transactions, compensations, and rollback tasks.

The theory of the workflow systems has contributed to current workflow or business process standards, such as BPEL [7], XPDL [8], BPMN [9], and others. Workflow systems are highly used to define enterprise systems and to implement complex models in businesses. There are many areas where workflow system are being used, including Web Service compositions, scientific workflows, image processing, collaboration applications, transaction-based applications, business models, manufacturing applications, and many others. The wide use of workflow systems requires standardization and the open source communities and commercial vendors have recently proposed many standards and research studies in order to improve and take the workflow system to the next level.

2.2 Business Process Modeling Notation (BPMN 2.0)

BPMN 2.0 is a standard from the Object Management Group (OMG) designed to model business processes in a graphical way as well as in term of their semantics. BPMN closes the gap between business analysts and developers, and solves the inter-operation between a business process and the human level tasks in a standard manner. BPMN is constructed based on workflow control [3], data [4], resource [5], and exceptions patterns [6]. The aim of the patterns is to delineate the fundamental requirements in business process modeling [3]. BPMN 2.0 defines four types of conformances: Process Modeling Conformance, Process Execution Conformance, BPEL Process Execution Conformance, and Choreography Modeling Conformance [9]. The focus in this research concentrates on Process Modeling Conformance and Process Execution Conformance.

A Business Process (BP) is a series of activities that are joined together with a business goal. A BP can be divided into a private process and a public process. The difference is that a public process interacts with the external participants whereas a private process only interacts with internal tasks. A BP according to the BPMN 2.0 standard has five types of elements: 1) Flow Objects: activities, events, and gateways. 2) Data Elements: data objects, input/output datasets, and data Stores. 3) Connecting Objects: sequence, messages, Associations, and data associations. 4) Swim-lanes: pools, lanes. 5) Artifacts: group, and annotations.

Flow Objects

Activities are a single work task performed within a BP. An Activity can be atomic or not, and acts as a super class for all the activities in the model. A task extends

from an Activity. There are different types of tasks: a service task represents a connection to a Web service and its invocation; a send task is used when the purpose of the task is to send a message to an external participant; a receive task waits for a message from an external participant; a user task represents a human intervention in the process as well as a manual task with the difference that the manual task is not controlled by the process engine; a business rule task integrates business rules into the BP; a script task represents a set of operations in a given scripting language and it is executed by the process engine.



Figure 2: BPMN 2.0 Tasks

Figure 2 shows the graphical representation of the seven types of tasks. A task can be marked with a loop operation, a multi-instance operation, or compensation operations.

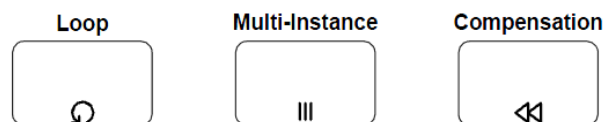


Figure 3: Tasks Markers [9]

Compensation is the process of undoing an unsuccessful execution of a task. It handles exceptions and transaction rollbacks. Compensation will always be triggered by a compensation event. Multi-instance creates as many instances of that task as specified in the task's properties whereas a loop executes the same instance of the task N times. Another type of Activity is a sub-process. A sub-process represents a process that is

embedded within another process with the purpose of reuse or organization. The sub-process can be represented as a collapsed or expanded view and the same tasks markers can be applied to it.

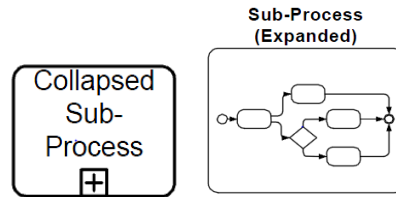



























































Figure 4: Sub-process [9]

An event extends from an Activity and is part of the Flow objects category. An event is something that happens during the execution of a process [9]. There are three main types of events: 1) Start event, which indicates the starting point of the process or sub-process; 2) Intermediate events, which indicate that something happens in the process during its execution 3) The End event, which indicates the end of a process. These three types can be either catching events or throwing events. Each type of events implements a series of definitions, which include: none, message, timer, error, escalation, cancel, compensation, conditional, link, terminate, and multiple. Table 1 summarizes the types of events in a BP and their graphical representation. Message events trigger or catch an event with a message attached to it, timer events trigger the start of a timer or catch an event when a timer expires, error events trigger or catch exceptions, escalation triggers the next step in the process, compensation triggers the next step in the process to compensate from error or rollback operations, cancel events terminate certain task operations, conditional events are triggered or caught when a condition is met. Link events connect tasks through events, terminate events stop the execution of the process, and multiple events create different instances of the event.

Table 1: BPMN 2.0 Event types [9]

Types	Start			Intermediate				End
	Top-Level	Event Sub-Process Interrupting	Event Sub-Process Non-Interrupting	Catching	Boundary Interrupting	Boundary Non-Interrupting	Throwing	
None								
Message								
Timer								
Error								
Escalation								
Cancel								
Compensation								
Conditional								
Link								
Signal								
Terminate								
Multiple								

Gateways in a business process give the flexibility of implementing most of the workflow patterns described by Aalst et al [3]. There are six special types of gateways:

- 1) exclusive gateways (XOR),
- 2) event-based gateways,
- 3) parallel event-based,
- 4) inclusive (OR),
- 5) complex (Multi Choice, multiple alternatives),
- and 6) parallel (AND).

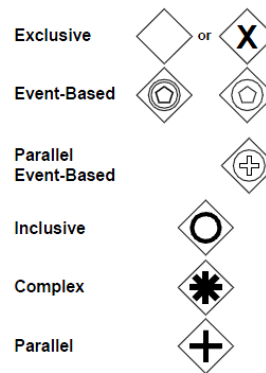


Figure 5: BPMN 2.0 gateways [9]

Data Objects

Data objects represent the data flow in the process. Even though BPMN 2.0 does not provide a specification for data flow, there are elements that represent explicitly (in a graphical representation) or implicitly (within properties in the flow objects) the data that is used within the business process. A data object is the single unit of data or collection of data in the BP. Figure 6 represents the data objects in a BP.



Figure 6: Single data object, Collections, Stores. Data Input/Output [9]

Data stores represent data that is persisted beyond the process execution. Another type of data items are properties. A property holds information about the process, flow objects, and other elements. It is used to describe the element and it does not have graphical representation. Data inputs and data outputs represent the data required to execute a task and its results respectively. Data objects can be associated with two tasks by an annotation element, or they can be associated directly to the connection between the tasks, as seen in Figure 7. The main difference is that the association within tasks

perform a data copy, whereas the data associated with the sequence element transfers the data from one task to another.

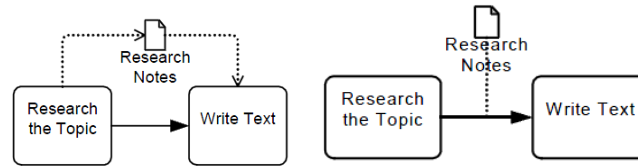


Figure 7: Connecting data to tasks. Connecting data to connecting elements [9]

Connecting objects, Pool and Lanes, and Other Artifacts

BPMN 2.0 has three types of connecting objects: a sequence connection, which connects activities explicitly; a message connection, which connects processes through messages; an association connection, which connects data objects and annotations.

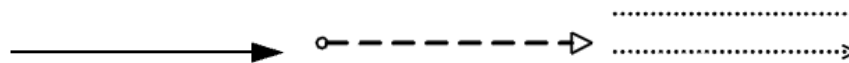


Figure 8: Sequence. Messages. Association

A pool is a graphical representation of a participant in a collaboration process. In a BP, a pool represents a single process. A Lane on the other hand represents a sub-partition within a process pool. Pools and lanes are used to graphically organize the process and its semantics which can be determined by a specific domain.

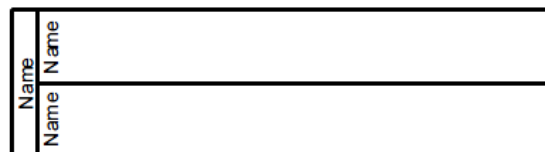


Figure 9: Pool with two lanes [9]

A group and an annotation are two additional elements that help to model a process without affecting its execution. These artifacts give a better understanding of the process semantics.

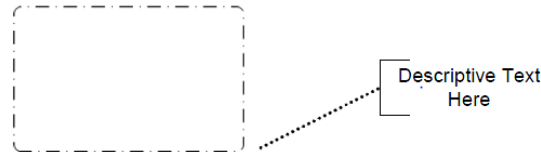


Figure 10: Group Artifact. Annotation Artifact [9]

2.3 Business Rules

Business process models are one essential part in a workflow within an organization. These models must be able to fully represent business decisions and constraints. The decision points in BPMN are represented in gateways and business rule tasks, but the actual implementation of the decision is left to the reasoning engine in this case the business rule engine. One of the main advantages of BPMN 2.0 is the integration with a business rules engine through its business rule tasks. This allows the model to be simplified and allows the business rule engine make complex decisions based on production rules. A business rule engine is a part of an expert knowledge-based system. A production rule is a basic *when <pattern> then <action>* structure that represents knowledge in a declarative way. These structures are passed through an inference engine that decides which actions to take.

In his paper Dr. Charles Forgy proposed the RETE algorithm [10], which implements an inference engine based on rules. This research was a great contribution for many of the current business rules engines that implement the RETE algorithm with

some extensions and optimizations. Rules are stored in what is called a production memory. Facts, which are part of the knowledge base, are stored in what is called working memory. The inference engine then takes the facts and matches them with the production memory in a forward chaining fashion. The process is divided into two steps 1) building the graph and 2) matching patterns.

The best way to explain the RETE algorithm is by example. Assuming the following rule is part of an emergency response system which is going to perform an assessment of the situation where the status of an emergency is a warning level and the population of the county where the emergency is taking place is more than 2000.

```
WHEN
    emergency.status == "Warning" AND county.population >2000
THEN
    doAssessmentOfSituation()
```

RETE converts the set of rules into a directed acyclic graph which contains five types of nodes: root node or rete node, entity nodes, alpha nodes, beta nodes, and terminal nodes. Entity nodes represent the kind of facts that are stored in the memory; this concept is the same as classes and objects in Java. In the example, there are two fact types: Emergency and County. Alpha nodes represent a literal condition evaluation. These nodes are also called 1-input nodes. In the example there are two alpha nodes, one attached to the entity Emergency and one attached to the Entity County. The alpha node attached to the Emergency entity evaluates the status of the emergency, in this case *status="Warning"*. The second alpha node attached to the County entity evaluates the population attribute, in this case *population>2000*. Beta nodes perform joins of two alpha nodes. These nodes are also call 2-input nodes. Beta nodes have a temporal beta memory

where tokens from the left hand side of the node are stored and wait for tokens from the right hand side to fire the condition and then continue to the following node. In the example, a beta node joins both alpha nodes from Emergency and County entities, in this case [status = “warning”, population>2000]. A terminal node stores the action to take when the node is reached. In the example *doAssessmentOfSituation()*. Figure 11 shows the final result of the graph.

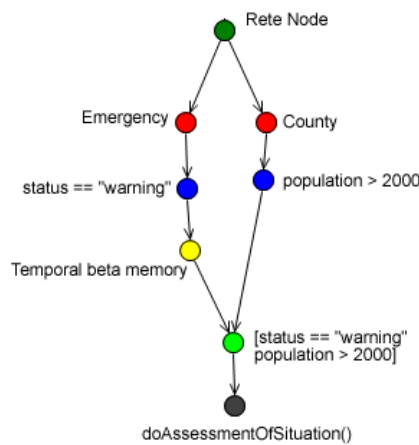


Figure 11: Rete graph of assessment situation rule.

Red nodes are Entity nodes, blue nodes are alpha nodes, light green is the beta node, yellow node represent the beta memory, and black node is a terminal node.

In the execution phase, the inference engine starts doing iterations through the facts and partially matching patterns based on the tree topology. If a fact is modified during the matching process, the process does another round of matching until there is no action to fire. The algorithm is time efficient but it is memory intensive.

Business rules have gained a lot of popularity in the last few years, and major enterprise systems have some implementation of this kind, including Oracle, IBM, JBoss,

and others. The main advantages of business rules systems are the ability to create rules in a declarative language, logic and data separation, speed of matching rules, integration with business process engines, and easy understanding and editing of rules. The implementation of business rules systems has triggered different research projects to extend the inference engine and make it more powerful such as temporal reasoning, complex events detection, domain specific languages, and many other adaptations to support specific problems in different domains.

2.4 Context Awareness in Business

One of the common problems in a business process model is the adaptability and context awareness. Business processes need a way to interact with the surrounding context and take actions according to the current situation. The context is defined as *“Any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves”* [11]. This definition can be extended in a business process as any information that can alter the normal flow of the process, including the business process itself. A good business process model is the one that is able to adapt to the current context.

The main components of the context for a business process are: 1) context query, 2) context Events, and 3) context services [12]. The context query module serves as a search engine of the context for the business process. For example in a bank transaction process, before the process validates a transaction, it may require to know if there are any crimes reported in the transaction’s location thus querying the context for any incidents in

the area. The context event module implements the way that the context interacts with the business process by publishing and responding to external events. The context can trigger events into the business process or the business process can register events to the context. For example, in the emergency system workflow the context needs to let the process know that an emergency is taking place in order to start the correct procedure to approach such emergency. The emergency process on the other hand can register events to the context about the status of the response. Context service module is the well-known context of the process. This module allows the context to register Web Services that will help the business process in certain tasks. For example, in an emergency system workflow the process will use weather services deployed in different locations to acquire information about the current temperature of certain locations. Figure 12 represents a context model using BPMN.

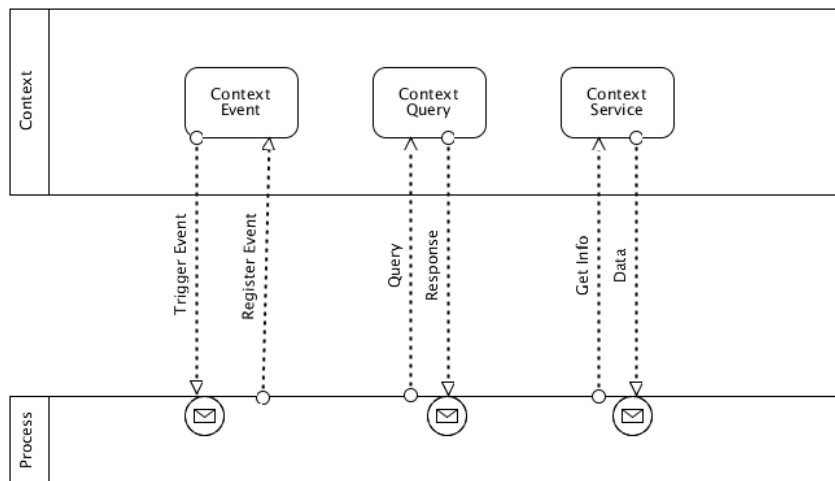


Figure 12: Context model

2.5 Geospatial Modeling

Geospatial analysis involves the design and execution of models in order to achieve a goal by converting geographic data into meaningful information that facilitates the decision making process [13]. Geospatial models are used in a broad variety of fields such as emergency systems, environmental planning and simulation, business intelligence, population growth analysis, and others. Geospatial analysis offers a unique characteristic which includes a visual aid to understand a situation on the Earth. A formal definition of the basics of geospatial analysis has been a subject of discussion for quite some time without a consensus in formalizing a broad definition for the different applications domains. Different research projects have addressed the issue of defining the core components in geospatial analysis, example being Map Algebra [14] and Universal Analytical GIS operations [15]. During the next paragraphs, basic Geospatial concepts will be described as it is necessary to understand the terminology and the base framework for this thesis. This definition is based on different approaches in research projects as well as a survey of daily use of the geospatial tools and their usability.

Location is defined as a pair of latitude and longitude coordinates that represent a point on the Earth. A point's latitude is the angle made in the North-South direction relative to the Equator line. Similarly, a point's longitude is the angle made in the East-West direction with the Prime Meridian Greenwich line. These coordinates are defined as un-projected coordinates. In this thesis, only un-projected coordinates are used for simplicity. For geometric operations the coordinates will be projected in Web Mercator, standard projection, which is used by most of the Map technologies, including Google Maps [16] and ArcGIS APIs [17]. There are two types of data used to represent

Geographic information: Vector data and Raster data. In this research the main concern will be operations over vector data, though similar assumptions can be made to extend the framework to work with raster data. Basic geometry in vector data are: Point, Line, and Polygon. A point is a single pair of coordinates. A line is two or more points joined together to form a line. A polygon is a set of coordinates joined together which ends at the same point that starts.



Figure 13: Point, Line, and Polygon

For this thesis, geospatial analysis is divided into four main categories of primitives operations. These operations are the result of the most common tasks used by different software vendors, including basic operations in Map Algebra. Table 2 describes the operations.

Table 2: Basic Geospatial Operations by Category

Geospatial operation category	Operations
Locational operation	<ul style="list-style-type: none"> - Distance - Geometric Relationships (Crosses, disjoins, intercepts, within, touches, and contains) - Buffer
Search Operations	<ul style="list-style-type: none"> - Spatial search or filter - Filter by attributes - Identify features
Network operations	<ul style="list-style-type: none"> - Point of references (northOf, southOf, eastOf, and westOf) - Driving time and driving distance
Measurement operations	<ul style="list-style-type: none"> - Area - Length - Elevation

Locational operations are well known operations in GIS. In this first group, there are three basic operations, distance between two geometries which can be applied to points, lines, and polygons. Geometric relationships are Boolean operators that relate two geometries with a specific relationship. Crosses evaluates to true when two lines have the same point in common except for the endpoints, a line and polygon have a at least one point in common and the line is not within the polygon. The crosses operator does not apply to point geometry. Disjoins returns true when the interception of two geometries including points, lines, and polygons is empty. Intercepts evaluates to true when the interception of two geometries is not empty. Within returns true when the base geometry is the interception of the comparison geometry. Touches evaluates to true when the interception of the geometries is not empty but the interception of their interiors is empty, only applies for lines and polygons. Contains returns true when the comparison geometry is within the base geometry. The buffer operation returns a polygon which represents a

base geometry expanded by a distance. The second group is the search operators. This group represents operations such as search, filter, and identify. These operations are used to perform spatial filter and retrieve information about features in the map. The third group is the network operations. This group performs operations of features in the map that belongs to the same network, for example roads, sewers, government buildings, energy lines, and others. The operations in the network group are divided in two: position in reference to another geometry (north, south, east, or west) and driving time and distance in the roads network. The last group is the measurements operations. This group has three main operations: area, length, and elevation. These four groups are the basic framework used in the following chapters.

Today GIS data is easily accessible from Web Services. Companies such as Google [16] and ESRI [17] are taking a huge step to offer GIS data and operations as standard Web Services. Also, many standards have been proposed by the Open Geospatial Consortium (OGC) whose main goal is to standardize the way applications interact with GIS services. The main goal of GIS Web Services is to give an easy interface for complex operations. These Web Services are usually developed in a RESTful [18] way and with SOAP [19] interfaces. There has also been great improvement in the open source community with Web Services standards such as Features Services [20], GML [21], Geo-processing services [22], and others. Web Services enable developers to build applications with geospatial awareness without worrying about the infrastructure and algorithmic complexity that some of the GIS tasks brings to the table. In this research, a number of Web Services are utilized in order to provide Geospatial context awareness to business processes. The assumption is that the

Web Services offer an optimal implementation of each operation which enables integration with business processes.

CHAPTER 3

GEOSPATIAL AWARENESS IN BUSINESS PROCESSES

3.1 Problem description

Modeling business processes has been a very important topic in the research community for the past few years. Web Services have enabled different protocols to create process compositions thereby simulating internal and external business processes and their interactions. Business Process Execution Language (BPEL) [7] is the most commonly used standard nowadays to compose web services, though its main limitation is interacting only with SOAP [19] Web services. In many cases a BPEL graphical representation does not provide a top level model for a business process, but instead provides a chain of Web Services. BPMN 2.0 [9] answers this problem with the ability to truly define a process in terms of business tasks which includes but is not limited to Web Service calls. As explained in Chapter 2, BPMN has a graphical way to represent a business process, and also offers the ability to convert models to an execution process in BPEL or BPMN engines. BPMN can easily be integrated with business rules engines, which provides a certain level of intelligence to the process. BPMN is becoming a very important standard for enterprise systems to achieve the goals of integration and business intelligence. On the other hand, BPMN has some limitations that need to be addressed in the coming years. One of the main limitations is the lack of definition for metadata to represent nonfunctional requirements in the business process. Nonfunctional

requirements describe constraints, qualities, and context of the business process to achieve the desired goal. These requirements are essential to simulate a real world process. A business process must be able to interact with the context and be able to adapt to external and internal events.

This thesis focusses on the geographic domain, and describes geospatial modeling at different levels of the process: business process constraints, business process reasoning, and the context for a business process. In the following sections, three specific problems are discussed: 1) geospatial requirements and constraints in business processes which describe constraints in the task level as well as in the process level, 2) geospatial reasoning through business rules, which allows the business process to achieve geospatial business intelligence and complex event detection based on geospatial patterns, and 3) geospatial context awareness. This last extension allows the business process to understand the context from a geographic perspective.

3.2 Motivating scenario

Throughout this thesis, an emergency response system example is being used to show the importance of geospatial context awareness in business processes. Emergency systems are a vital component in government agencies and businesses to address situations that need immediate response, such as tornados, hurricanes, fires, floods, and others. Emergency systems are relevant to this thesis due to their unique characteristics. Emergency systems entail collaboration and real time data processing. There is a need to have a structured model to act in an emergency, enabling collaboration among different actors which are in this case, government agencies, responders, and victims of the

emergency, and others. Emergency systems workflows have very different constraints that need to be added to the process model. One of these constraints is geospatial or location awareness imposed by the context of the emergency. Also, there is a need for intelligent systems that can propose solutions taking into account the actual facts or events in certain situation. For example, when a fire emergency is detected in a building structure an emergency response process is started to address the situation.

- A quick assessment is performed and the first responders are notified, in this case the closest firefighters, the closest hospitals, and the police officers that are nearby the building.
- It is also important to analyze factors such as nearby buildings, schools, and other critical facilities that are close to the location of the fire and can also be affected; also, how many people are in the area and possible evacuations zones.
- The next steps might be to escalate the awareness to the community that is in potential danger; monitor social networks to detect events that can be related to the fire's location; notify government agencies to start assessment in different areas such as energy resources, transportation resources, historic resources, health issues, and others.

In summary, the emergency response system must be able to adapt its procedures, models, and response according of what is currently happening in the context of the event. There are different actors involved in the emergency, and the context of the system has many constraints that need to be addressed. In the framework of an emergency, this thesis concentrates on the internal process of the emergency system which includes:

assessments, resource evaluation, infrastructure evaluation, operational plan, and response plan. Different considerations are described for internal and external interaction of the context with each part of the process. Use cases will be presented and described using BPMN, business rules, event detection, and the context of the system using the geospatial domain. Figure 14 shows an emergency response system expressed in BPMN 2.0.

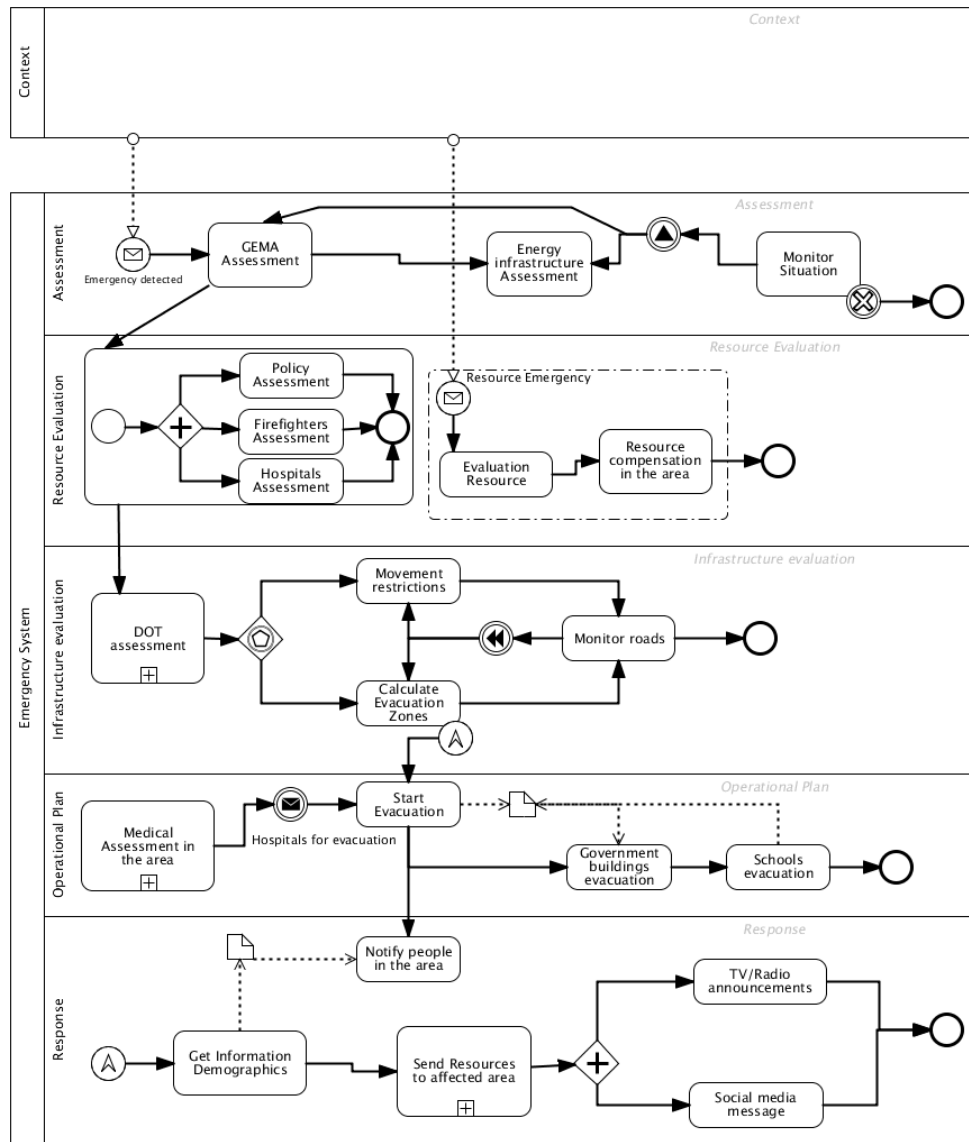


Figure 14: Emergency Management Business Process

CHAPTER 4

RELATED WORK

In recent years, the research in business modeling has been very active. The main goal in these studies is to be able to model enterprise systems that represent real life scenarios as closely as possible. Business process and Business rules systems are part of many commercial and open source projects which offer constructs to represent business operations and constraints through many different methods [23]. A business process can be classified into different types according to their focal modeling constructs. Classifications include activity centered, object centered, and resource centered. Business rules are also classified as rules to express constraints, conditions, reaction rules (Event-Condition-Action), production rules, and transformation rules [23]. The integration of a business process and business rules enables coordination of workflow and tasks among the enterprise systems [24]. Business process modeling has evolved from Petri-nets theory [2] and workflow patterns [3] to full specifications such as BPMN 2.0 which has a complete set of artifacts as well as execution semantics to represent complex business processes [9]. Another aspect of business process modeling involves accurate capture of operational behavior and associated constraints in the process [25]. These behaviors and constraints include performance expectations, policy, constraints, and security which are characterized in the system as non-functional requirements into the business process [25]. BPMN does not provide a ready support for expression of non-functional requirements, but different studies have approached these issues by proposing extensions to BPMN to

add constraints into the model and execution of the process. Business rules provide artificial intelligence to enterprise systems through production rules reasoning and acting according to the constraints of a business. Business rules have also been extended to support complex event detection [26] and to support temporal constraints [27]. In order to achieve the ultimate goal of fully describing a business process as similar as possible to real life processes, studies have focused on the context of the process and describing the different constraints and interactions with the environments that affect the normal flow in a business process [12]. Contextual applications enable business intelligence and therefore make the process more flexible and adaptable to different circumstances of the environment [11]. An ideal business process is one that describes a standard model, provides inference based on business facts, and is able to adapt to the context in which it runs.

On the other hand, Geographic Information Systems have gained a lot of momentum in the Web. Different commercial and open source initiatives have been working towards the full specification and standards for Geographic Web Services. The Open Geospatial Consortium (OGC) has released open source standards to offer geographic data and operations through the Web such as Features Services (WFS) [20], Web Geo-Processing Services (WPS) [22], Geospatial Markup Language (GML) [22] and others. GIS leading companies such as ESRI and Google have also released their own version of RESTful [18] APIS and SOAP [19] Web Services to offer different GIS operations in the Web. Today, it is very common and easy to find Web Services that provide basic and even complex GIS operations that are easily accessible and straight forward to integrate to any application. One of the benefits of Web Services is certainly

the capability to create complex chaining operations and generate more complex services for businesses. OGC has been working in the last years to integrate their standards towards Web Service compositions. They explore the use of BPEL and BPMN as valid options to create geographic web services chaining [28]. Other studies have approached the problem by creating their own workflow standard such as OGC-ORCHESTRA [29]. ESRI has as one of their main products the Model Builder tools that allow the composition of geo processes using the Python programming language and integration with many data sources including geo databases, ArcGIS web services, and ArcGIS processes [30]. Another study has shown a comparison of using BPEL and BPMN as main engines to compose GIS services [31]. In that study, both technologies have been shown to enable integration between any GIS service. The main idea when using BPEL is to have a WSDL document for each GIS service. With BPMN the idea is even simpler since BPMN supports services and custom implementations of tasks. One of the recent studies, Geo-Ontology tools: the missing link [32], expresses the need to include geospatial information into business processes using ontologies and models such as BPMN to standardize the way that processes to talk to each other in geospatial language.

The development in the area of business processes, including modeling and reasoning with business rules and geospatial integration into workflow systems, need to be generalized in a way that geospatial information can be easily integrated into business processes. In this research, and based on previous studies, the approach is to fully describe a system that supports geospatial requirements into business process. In this way, geospatial requirements can be integrated into the business model itself, creating a truly geospatial business intelligence. The approach goes beyond simply creating GIS

Web Services chaining by fully describing a business process with its three main components, which are the model, the inference, and the context based on geospatial requirements.

CHAPTER 5

GEOSPATIAL MODELING IN BUSINESS PROCESSES

Presented in the next sections are the three main contributions of this thesis. These contributions are: 1) a geospatial constraints language for BPMN, 2) an extension for business rules to match geospatial patterns, and 3) a description of the external geospatial context for a business process.

Constraints within a business process have been a subject of study for several years. Most of these studies, specifically in BPEL and BPMN, focus on embedding constraints and actions within the specification by using events and alternative tasks in order to respond to validations within the execution of the process. Our approach abstracts the constraints and validations and converts them into a specific language and implementation, in this case focusing on the geospatial domain. This language is extensible, modular, and self-contained, and provides a powerful way to define geospatial constraints within a business process. The second contribution extends the business rules engine to support geospatial patterns. Our extension allows the business rules engine to match facts based on geo-location, and it specifies a language to write geospatial rules. The third contribution is the description of the geospatial external context of a business process. Our extension allows the business process to query, interact, and gather information of the context with geospatial requirements. These three contributions are validated using a prototype demo in Chapter 7. The prototype is an emergency response example that uses each of these extensions to enrich the business process execution.

5.1 GCL: Geospatial Constraint Language for BPMN

As the problem and motivation states in the previous chapter, this research extends BPMN to support geospatial awareness within the business process. In order to achieve this goal, basic geospatial operations defined in chapter 2 are integrated with artifacts in the business process conformance. With this extension a business analyst will be able to write geospatial constraints and define a graphical representation of a business process model that requires location awareness.

Five different patterns have been identified that describe the geospatial relationships in a business process. These patterns are described using OCL-like syntax [33]. OCL is a language created by the Object Management Group (OMG), which describes constraints for Unified Modeling Language (UML) diagrams. For this thesis, we have created a language for specifying geospatial constraints called Geospatial Constraint Language (GCL). These constraints are usually nonfunctional requirements that are not possible to include in the process model. OCL-like syntax has been chosen due its straightforward approach to writing constraints and extending a model without altering its execution or complicating the model itself. There is no standard to define nonfunctional requirements in business processes, but OCL can be easily extended to apply to artifacts in the model. Definition 1 formalizes the concept of GCL. Definition 2 formalizes the concept of the task and how the metadata information must be attached to the task in a geometry property.

Definition 1: The Activity context is made up of the different constraints that need to be added to the business process in order to execute and simulate the behavior in real use

cases. Geospatial constraints are one special type of requirements. These constraints are expressed using GCL (Geospatial Constraint Language).

Definition 2: Each Task has a *Geometry Property*, where Geometry is a super class of a Point, Line, and Polygon and is represented as T.Geometry. This Geometry is referred to as location. *The Geometry property* for a Task can be attached in designing time or execution time of the business process as a property or parameter of the task.

GCL constraints are applied at the task level and can be validated as preconditions, postconditions, or invariants. A precondition is a constraint that is verified before the execution of the task. A postcondition is a constraint that is applied after the execution of the task. An invariant validation ensures that the property or parameter geometry does not change during the execution of the task. Another important property of GCL validation is the *reachability property* in execution time. When a constraint is applied and involves at least two tasks, the constraint will be validated when the last task involving the constraint is reachable in the process instance, as defined in Definition 3.

Definition 3 - Reachability property: Let us assume there are two tasks T_i and T_j that are part of a Workflow W . T_i and T_j have a geospatial constraint $GCL(T_i, T_j)$ which will be executed only when there is an execution path between T_i and T_j , and both tasks have been reached.

BPMN 2.0 defines seven different types of tasks. The semantics for GCL validation of the constraints vary between tasks types.

- **Manual Task and Human Task:** The semantics in this case applies to the actor's location that performs the task. The precondition, postcondition, and invariant are validated against the location information of the actor.
- **Service Task:** The semantics in this task applies to its input and output datasets which must contain location information in order to perform the validation. If the GCL is a precondition of the task, then the validation is performed in the input parameters of the Web Service, on the other hand if it is postcondition then the validation is performed in the output of the Web Service.
- **Send Task and Receive task:** In a send task the validation is performed before sending the message, the assumption is that the message contains location information and the constraint validates whether the message's location meets the GCL constraint. Analogous, in a receive task the constraint is validated after the message is received and the assumption is that the message has location information, which indicates the origin of the message.
- **Script task:** The validation is applied to the input or output parameter of the script task. The location information of the task may change when executing the script therefore enabling post conditions validations.
- **Business rule task:** GCL validation does not apply in this case. For more details on geospatial rules reasoning see Chapter 5.2.

In the next sections, the five GCL patterns are presented with a full example demonstrating their theoretical validation using an Emergency System workflow.

Proximity relationships

G1. Proximity relationships	
Problem	In a business process two or more tasks need to be executed or placed in a specific location in proximity relationship with other tasks.
Solution	<p><u>Proximity constraint:</u> Context T_i, T_j: $\langle pre/post/inv \rangle: Distance(T_i.Geometry, T_j.Geometry) < R^+$</p> <p><u>Distant constraint:</u> Context T_i, T_j: $\langle pre/post/inv \rangle: Distance(T_i.Geometry, T_j.Geometry) > R^+$</p> <p><u>Same Place constraint:</u> Context T_i, T_j: $\langle pre/post/inv \rangle: Distance(T_i.Geometry, T_j.Geometry) = 0$</p> <p>Where R^+ is a positive real number.</p>
Examples	<ul style="list-style-type: none"> - In an emergency system workflow, designed to respond to a fire. The closest firefighter station that is nearby, at most 5km, must respond to the emergency. <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> Context SendAmbulance, TakeToHospital: inv: Distance(Fire.Geometry, FireFighterStation.Geometry) < 5km </div> <ul style="list-style-type: none"> - In a backup process, it is recommended to perform a backup task in different locations with considerable distance to lower the probability of losing data due to an emergency in one of the facilities. <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> Context Backup1, Backup2: inv: Distance(Backup1.Geometry, Backup2.Geometry) > 10km </div>

Geometry task relationships

G2. Geometric task's relationships	
Problem	Geometry correlation among activities in execution and placement. Validation of constraint based on geometry relationships in the process.

Solution	<p><i>Context T_i, T_j:</i> $\langle pre/post/inv \rangle: T_i.Geometry \langle relation \rangle T_j.Geometry$</p> <p>Where relation is one of the following: crosses, disjoint, intercepts, within, touches, and contains. These operations return Boolean values.</p> <p><i>For example:</i> <i>Context $T1, T2$:</i> <i>pre: $T1.Geometry$ within $T2.Geometry$</i></p>
Examples	<ul style="list-style-type: none"> - In an Emergency system workflow for a tornado emergency, a School Evacuation task must evacuate schools that are within the tornado path. <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Context TornadoDetection, NotifyCommunity: inv: School.Geometry within TornadoDetection.Geometry</p> </div>

Task Measurement relationships

G3. Task Measurements relationships	
Problem	In a particular process, activities have restrictions that are related with the area, length, and elevation of its related Geometry.
Solution	<p><i>Context T_i:</i> $\langle pre/post/inv \rangle: area(T_i.Geometry) < R^+$ $\langle pre/post/inv \rangle: area(T_i.Geometry) > R^+$ $\langle pre/post/inv \rangle: area(T_i.Geometry) = R^+$</p> <p><i>Context T_i inv:</i> $\langle pre/post/inv \rangle: length(T_i.Geometry) < R^+$ $\langle pre/post/inv \rangle: length(T_i.Geometry) > R^+$ $\langle pre/post/inv \rangle: length(T_i.Geometry) = R^+$</p> <p><i>Context T_i inv:</i> $\langle pre/post/inv \rangle: elevation(T_i.Geometry) < R^+$ $\langle pre/post/inv \rangle: elevation(T_i.Geometry) > R^+$ $\langle pre/post/inv \rangle: elevation(T_i.Geometry) = R^+$</p> <p>Where R^+ is a positive real number.</p>
Examples	<ul style="list-style-type: none"> - In emergency system workflow, the assessment of the situation must be performed in an area of at least 10km². <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Context AssessmentSituation: inv: area(AssessmentSituation.Geometry) > 10km</p> </div> <ul style="list-style-type: none"> - The maximum length of an evacuation path is 5km <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Context Evacuation: inv: length(Evacuation.Geometry) < 5km</p> </div>

Geographic Network operations

G4. Task Network operations	
Problem	Correlation of tasks according to operation within the network that the task's geometry is a part of.
Solution	<p><i>Context T_i, T_j:</i></p> <p>$\langle pre/post/inv \rangle: T_i.Geometry \text{ NorthOf } T_j.Geometry$ $\langle pre/post/inv \rangle: T_i.Geometry \text{ SouthOf } T_j.Geometry$ $\langle pre/post/inv \rangle: T_i.Geometry \text{ EastOf } T_j.Geometry$ $\langle pre/post/inv \rangle: T_i.Geometry \text{ WestOf } T_j.Geometry$</p> <p><i>Context T_i, T_j:</i></p> <p>$\langle pre/post/inv \rangle: \text{Route}(T_i.Geometry, T_j.Geometry).drivingTime < R^+$ $\langle pre/post/inv \rangle: \text{Route}(T_i.Geometry, T_j.Geometry).drivingDistance < R^+$</p> <p>Where R^+ is a positive real number.</p>
Examples	<ul style="list-style-type: none"> - In an Emergency System Workflow, the driving time of sending people to nearby hospital must be less than 5 minutes.
	<p>Context HospitalTask, Evacuation:</p> <p>post: $\text{Route}(\text{HospitalTask.Geometry}, \text{Evacuation.Geometry}).drivingTime < 5min$</p>
	<ul style="list-style-type: none"> - If there is a tornado in south Atlanta then an warning level must be applied only to the south of the state <p>Context TornadoDetection, CountiesSouthNotification:</p> <p>inv: $\text{County.Geometry southOf TornadoDetection.Geometry}$</p>

Geospatial Filter relationships

G5. Search and Filter relationships	
Problem	Correlation of tasks in a fixed location.
Solution	<p><i>Context T_i:</i></p> <p>$\langle pre/post/inv \rangle: T_i.Geometry \text{ IN Filter}(POLYGON)$</p> <p>Polygon is a constant input. The constraints validates whether the task is within a specific fixed area.</p>
Examples	<ul style="list-style-type: none"> - In a bank loan process, the location of a transaction must be within the US territory to be valid. <p>Context Transaction:</p> <p>inv: $\text{Transaction.Geometry IN filter}(\text{USA.Geometry})$</p>

Graphical representation of geospatial constraints

GCL requirements can be applied in different level of the process. To express this constraint in a graphical model, a geo-marker can be attached to different artifacts of the process. This marker has the shape of the Earth as seen in Figure 15 and it represents location information for the artifact. For example, when a task is marked with a geo-marker it means that the task has a location information and that it can potentially be validated with GCL constraints. The marker can be applied to different artifacts, such as annotation artifacts, grouping artifact, and pool or lanes as seen in Figure 16. These artifacts represent a GCL constraint relationship among tasks.



Figure 15: Geo-Marker. BPMN with two tasks marked with geospatial data

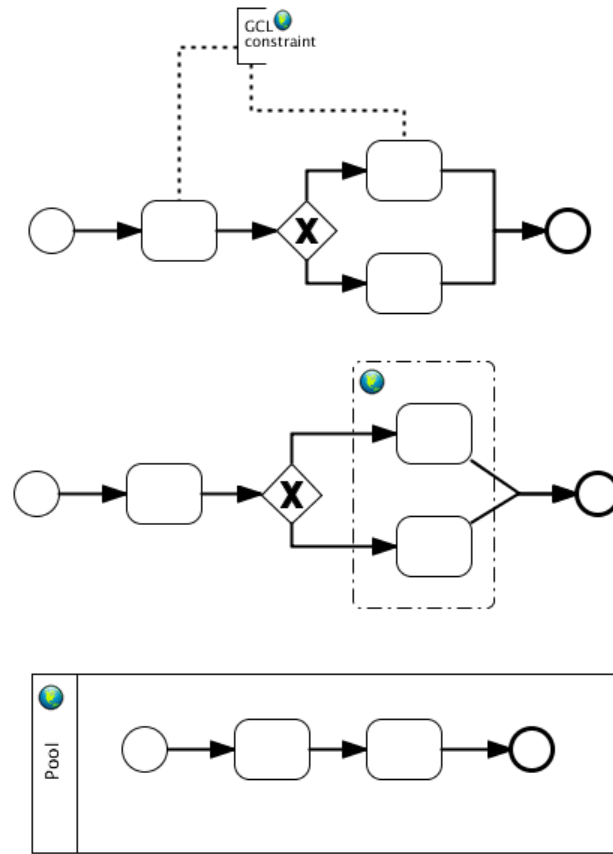


Figure 16: Graphical representation of BPMN with Geo markers

As stated in the BPMN 2.0 document by OMG in the extensibility section [8], the model can be extended with attributes and properties without altering the semantics of any BPMN artifacts. As explain before, geometry information for each task is attached to the properties attribute or parameter input sets. The constraints document can be attached as part of an extension. Though in the prototype demo for this thesis in Chapter 7 another solution is given, where the GCL document is a single file containing the constraints for a BPMN document. Table 3 shows the possible XML structure of a GCL constraint.

Table 3: GCL constraints extension for BPMN

BPMN Schema extension
<pre> <?xml version="1.0"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> <xs:element name="gclConstraint"> <xs:attribute name="constraint" type="xs:string"/> <xs:complexType> <xs:element name="tasks"> <xs:sequence> <xs:element name="task" type="xs:int" /> </xs:sequence> </xs:element> <xs:sequence> <xs:element name="param" type="xs:string"> <xs:attribute name="name" type="xs:string"/> </xs:element> </xs:sequence> </xs:complexType> </xs:element> </xs:schema> </pre>
Example
<pre> <bpmn:extension mustUnderstand="true" definition="bpmn:gclConstraint" /> ... <bpmn:task name="Task1" id="ID_1"> <bpmn:gclConstraint> <gclConstraint:gclConstraint constraint="closeby"> <gclconstraint:tasks> </gclConstraint:task>1</gclConstraint:task> </gclConstraint:task>2</gclConstraint:task> </gclconstraint:tasks> <gclConstraint:param name="distance">9000</gclConstraint:param> </gclConstraint:validate> </bpmn:gclConstraint> </bpmn:task> <bpmn:task name="Task2" id="ID_2"> </bpmn:task> </pre>

GCL validations

The validation of each constraint returns true or false only if the tasks involved in the validation have been reached at execution time. This follows the reachability property explained before. If the tasks involved in the validation are not yet reached, then the validation is omitted until the last task is about to be executed. After the validation is performed, the process engine must take a valid action within the process execution to respond to the validation. For this reason, the process engine throws an exception event. The action to take when this event is triggered is left to the business process designer. There are different options based on the BPMN specification:

- Compensation: If the task returns false, then the task can trigger a compensation event to perform an action to change the normal execution of the process flow.
- Rollback: If the process implements transaction tasks, the process can roll-back the execution of each task involved in the constraint validation.
- Trigger event: If the validation succeeds or fails, the process can trigger an event which handles the current state of the process.
- Force the constraint: If the constraint validation returns false, the process can decide to wait until the condition is met.
- Terminate the process: If the constraint validation fails, the process can terminate the execution.

Emergency System Business Process and Geospatial constraints example

The following example describes an emergency management system business process. The process is divided into five lanes: assessment, resource evaluation, infrastructure evaluation, operational plan, and response. These five lanes correspond to a standard practice of approaching an emergency. The interaction with the context is done by a collaboration process between the context pool and the emergency pool. The process starts its operation when an emergency is detected. The context triggers an event to the emergency respond process through an event message. This message contains the location information of the emergency and activates the assessment lane. Figure 14 illustrates the business process. When an Emergency is detected, the first task GEMA_Assessment is activated, getting as input the information from the context about the event. The assessment is a key part of the emergency process since it determines the level of emergency of the process. A geospatial constraint can be applied in this level of the process: GEMA_Assessment must be performed within 5km of the emergency detected (Event). This corresponds to the pattern G2. A geospatial operation was used to buffer the geometry of the event detected by 5km and then the task is validated with the relation *within*.

Context E, T inv:

T.Geometry within Buffer (E.Geometry, 5km)

This example illustrates the ability to use other geospatial operations within the pattern validations, in this case a buffer which is a basic operation to filter features in a map by a given distance. In the following constraint, the context is defined in the lane level and the OCL operator *forAll* is used to iterate through the tasks that belong to the

Assessment lane. This shows that other OCL operators can be used to construct complex geospatial validation among tasks using the pattern G1.

Context AssessmentLane inv:

$$distance(Self.T1, self.T2) < 5km \rightarrow \text{ForAll}(T1, T2 \in \text{Assessment Lane})$$

One critical step in the Emergency process is the calculation of evacuation zones. For this to be optimal, a constraint can be added to guarantee that the path is optimal in driving time. This illustrates the pattern G4.

Context CalculateEvacuationZones inv:

$$route(Event.Geometry, SafePlace.Geometry).drivingTime \leq OptimalTime$$

Another constraint is the distance of tasks in the assessment lane must be at most 5km. The assessment lane has a task that monitors the situation while the process is executing, this monitoring must be performed in the same area where the initial assessment was performed to maintain the execution of the process focus in one area only. The following constraint shows the pattern G5.

Context MonitorSituation inv:

$$Self.Geometry \text{ over } Filter(Buffer(Event.Geometry, 5km))$$

Another important assessment in the process is the sub-process DOT assessment which assesses the status of the roads and transportation system. In order to be optimal, this assessment must be performed in an area of at least $10km^2$. This constraint can be expressed in the sub-process level with the pattern G3.

Context DOTAssesment inv:

$$area(self.Geometry) \geq 10km^2$$

As one can see in the previous example, several constraints and metadata can be added to the business process in order to enforce its execution according to the real life scenarios.

This proposal offers the opportunity for designers and implementers to define explicitly geospatial constraints and attached them to the process. The business process in the previous example uses artifacts that can be used to attach constraints metadata to the model, but more importantly than showing these constraints in the model is the ability for the process virtual machine to understand these constraints and enforce them.

5.2 GDL: Geospatial Domain Language for Geospatial Business Rules

GIS applications are an excellent tool to visualize a map and to overlay geographic data in order to provide tools to help human-made decisions in day-to-day situations. The next generation of GIS applications must not only be able to display data but be able to reason and propose concrete actions based on geospatial knowledge. Currently, there are many devices that record location information such as GPS devices, mobile phones, tablets, Web Services, sensors, and others. An optimal enterprise system for a business process is one that is able to integrate this information in real time. Business rules are a suitable component to achieve full integration between geospatial information and enterprise systems.

In this research, we proposed to extend business rules to be able to use geospatial correlations to analyze facts and events and make decisions in a business processes easier. Geospatial correlations are the different relationships among geographic datasets. In this work, the basic unit of processing is a *GeoFact*, which is an extension of a *Fact* in the knowledge base. A *GeoFact* has a location information. A *GeoFact* can also be considered a *GeoEvent* if it has a timestamp, as well as location information. A *GeoEvent* is defined as an *Incident* in the map. In this research the focus will be geospatial

correlations. The extension can be used with temporal reasoning to enable geo-temporal reasoning.

Definition 4: A *GeoFact* is an extension of a Fact in the knowledge base that has the property of Geometry which represents location information.

Definition 5: A *GeoEvent* is a special kind of GeoFact, which contains a timestamp in the knowledge base. A GeoEvent is considered an Incident.

GeoFacts semantics:

- GeoFact correlations use basic geospatial operations integrated with a rules engine.
- GeoFacts must have a geometry property. The Geometry can be a Point, Line, or Polygon, and it represents location information
- The engine must be capable to perform operations in different coordinates systems and projections.
- Correlation operations are represented as beta nodes in the RETE network, thereby not altering the RETE algorithm and allowing the engine to integrate geospatial rules with other kinds of rules.
- Since a GeoFact has a geometry property, different geospatial operations can be applied to generated new geometries “on the fly”, without altering the normal matching process in the rule engine.

In the following sections geospatial reasoning will be presented and divided into four categories that represent the basic units of analysis in GIS. The last section presents an example using the Emergency system response workflow.

Proximity rules

A proximity evaluator correlates facts when matching the current GeoFact to other GeoFacts that are nearby, distant, between distances, or in the same place.

Proximity operators:

GeoFact1 (this closeby [R^+ distance unit] GeoFact2)

<pre> rule when GeoFact1(this closeby [5km] GeoFact2) then Action... </pre>
--

In the previous example all the facts that are close by less than 5km to GeoFact1 will be matched. The rule uses the distance geospatial operator and the relationship is $distance(GeoFact1, GeoFact2) \leq 5km$.

GeoFact1 (this distant [R^+ distance unit] GeoFact2)

<pre> rule when GeoFact1(this distant [2km] GeoFact2) then Action... </pre>
--

The previous rule will match all the facts that are distant by 2km to the GeoFact1. The rule uses the distance geospatial operator and the relationship is $distance(GeoFact1, GeoFact2) \geq 2km$.

GeoFact1 (this sameplaceas GeoFact2]

<pre> rule when GeoFact1(this sameplaceas GeoFact2) then Action... </pre>
--

The previous rule will match all the facts that have the same location as GeoFact1. The rule uses the distance geospatial operator and the relationship is $distance(GeoFact1, GeoFact) = 0$.

GeoFact1 (this between [R^+ distance_unit, R^+ distance_unit] GeoFact2)

<pre> rule when GeoFact1(this between [1km, 10km] GeoFact2) then Action... </pre>
--

The previous rule will match all the facts that are between the distance range of GeoFact1. The rule uses the distance geospatial operator and the relationship is $1km \leq distance(GeoFact1, GeoFact2) \leq 10km$.

Geometry relationships rules

A geometry-relationship evaluator correlates facts based on their geometric relationships. Geometry relationships are Boolean operators such as crosses, disjoint, intersects, within, touches, and contains. These geometric operators are explained in detail in Chapter 2.

Geometry relation operators:

GeoFact1 (this crosses GeoFact2]

GeoFact1 (this disjoint GeoFact2]

GeoFact1 (this intersects GeoFact2]

GeoFact1 (this within GeoFact2]

GeoFact1 (this touches GeoFact2]

GeoFact1 (this contains GeoFact2]

rule
when
GeoFact1(this within GeoFact2)
then
Action...

The previous rule will match all the facts that evaluate true in the geometry relationship operation to GeoFact1, in this case *within*. The rule uses the relation operator and the relationship is *within(GeoFact1.Geometry, GeoFact2.Geometry) = true/false*.

Network relationship rules

A Network relationship correlates facts that are part of the same network. Facts such as roads, houses, highways, or sewers, for example, are part of the geospatial

network. In this thesis, the focus is on two basic operations: positioning based on a point of reference, and driving time and distance between facts.

Network operators:

GeoFact1 (this northOf < [distance]> GeoFact2]

GeoFact1 (this southOf< [distance]> GeoFact2]

GeoFact1 (this eastOf< [distance]> GeoFact2]

GeoFact1 (this westOf< [distance]> GeoFact2]

```
rule
  when
    GeoFact1( this northOf [5km] GeoFact2)
  then
    Action...
```

```
rule
  when
    GeoFact1( this southOf GeoFact2)
  then
    Action...
```

In the first example, the rule will match all the facts that are north by 5km of GeoFact1. The rule combines the distance operation with the location based on the point of reference of GeoFact1. The rules implement these two operations using the relationship: *distance(GeoFact1, GeoFact2)<=5km AND northOf(GeoFact1, GeoFact2) = true/false*. In the second example, the rule matches all the facts that are south of GeoFact1.

GeoFact1 (this drivingTime [time range] GeoFact2]

GeoFact1 (this drivingDistance [distance range] GeoFact2]

```

rule
  when
    GeoFact1( this drivingTime [ 1h 3h] GeoFact2)
  then
    Action...

```

In the previous example, the rule matches all the facts for which driving time is between one hour and three hours.

Spatial grouping rules

Grouping correlations evaluate facts that are within a set which has been spatially filtered by geometry, usually a polygon, or/and search query.

GeoFact1 over area:filter(bounding_box_polygon)

```

rule
  when
    GeoFact1 (this over area:filter(Polygon))
  then
    Action...

```

In the previous example, the rule matches if the GeoFact1 is over the set returned by the filter operation. This rule uses the filter operation and the relationship is *GeoFact1* \in *filter(Polygon)*. The operation returns true or false.

GeoFact1 over area:search(query, bounding_box_polygon)

```

rule
  when
    GeoFact1(this over area:search("Hospitals", Polygon) )
  then
    Action...

```

The example above shows a rule that performs an action if GeoFact1 is over the set returned by the search operation. This rule uses the search operation which has as parameters a query and polygon filter, and the relationship is $GeoFact1 \in search(query, Polygon)$. The operation returns true or false.

Emergency system response and geospatial reasoning

Table 4: Business rules in an Emergency system workflow

rule “Notify hospital and elevate the emergency status”
when
emergencyEvent(this closeby(0.5mi) hospital)
then
notifyHospital(hospital.id);
emergencyEvent.status = HIGH
rule “Notify government agencies in the county of the emergency”
when
county(this contains emergencyEvent)
then
county.notifyAgencies()
rule “Close roads that intercepts with emergency event”
when
road(this intercepts emergencyEvent)
then
road.close = true
rule “Assign hospital to a building for evacuation purposes”
when
building(this drivingTime [1min 5min] hospital)
building(this closeby [1mi] emergencyEvent)
then
building.evacuationHospital = hospital

In an emergency system workflow, there are several tasks that invoke the business rules engine in order to continue the normal execution. The business rules engine for this kind of workflow helps the decision process, given an emergency. For example, in the Assessment task there are several rules that can be applied at the rules engine level to determine different actions to take in case of an emergency. Table 4 describes the rules

that the assessment business rule will go through when determining those actions. The example assumes that there are GeoFacts of type Emergency Event (Point), Hospitals (Points), Schools (Points), Roads (Lines), and County (Polygon) in the knowledge base.

5.3 Geospatial Context for Business Processes

The context of a business process is usually very broad and hard to model. That is why the problem must be divided into specific subdomains in order to be able to describe it. Geospatial context is any information with location information that is relevant to the business process. The context for a business process is divided into three main components 1) context query, 2) context Events, and 3) context services. In this thesis, three geospatial extensions describe the context layer with geospatial awareness in a business process.

The first extension is a *Geospatial context query* module. This module allows the business process to query the context with geospatial queries. The basic operations in this module are filter, and spatial search. A filter operation allows the business process to ask questions based on geometric boundaries. In order to implement this, the context engine must be able to index location information to support queries such as: *What features are in a 5km radius from a location x,y ? What features are within the polygon(x,y)? What features cross the main highway in a certain location?* Along with any other queries that contain geospatial information. The search operation combines a spatial filter with other attributes in the context definition. For example, *what hospitals are nearby the location x,y and have availability? Or what are closest police officers from the location x,y ?* The

geospatial query engine must be able to parse the query and respond in a standard format for the business process to integrate this information within its activities.

The second extension is the *Geospatial events handler and Geospatial event publisher*. The context layer must be able to process any events that are relevant to the business process. This module can be implemented using a business rules engine with complex event detection and geospatial reasoning support. The engine must support different data input formats, such as KML, GeoRSS, GeoJSON, GML, ESRI's RESTful API [18], sensors, GPS devices, and others. It must be able to convert the data into meaningful artifacts for the business process, in this case GeoEvents or Incidents as explained in the previous section. Through the business rules engine, the context is able to invoke event handlers in the business process. In order to publish events, the business process registers in the context layer geo-events that might be relevant to other business processes. The context must be able to parse these events, and publish them in a standard format. Geospatial events handlers and publishing are part of the Context events module.

The third extension is the *Geospatial context services*. This extension allows the engine to register Geospatial Web services in order to be able to invoke them when needed by the business process. This layer serves as a Web Service catalog for the business process. These Web Services are different from the service task in the business process, since they are added to collect information about the context, rather than being part of the process itself. For example, a service task can invoke a GeoRSS service to know the current earthquake incidents, and then invoke a web service that will perform some action within the business process, if there are any incidents. The invocation of the context web services is not included in the process model explicitly.

These three extensions allow the context layer to understand geospatial information and serve as middleware for the business process and the external environment. The business process is then able to implement processes with adaptation capabilities and be more flexible according to the current status of the context. The context is generic enough and can be used in the same way for any use case, including the example of the Emergency system workflow.

CHAPTER 6

GEOFLOW: SYSTEM ARCHITECTURE

In the following sections, we present an overview of the system and a prototype implementation which validates the concepts explained in previous chapters. This prototype system is called GEOFlow. The Geospatial Context Awareness Business Process system is divided into three layers: 1) geospatial Engine layer, 2) business process engine and Business rules engine layer, and 3) context Awareness layer. These three layers communicate through API calls or Web Services calls. This allows each layer to act independently from each other and only utilize the other layers when needed. Figure 17 shows an overview of the system.

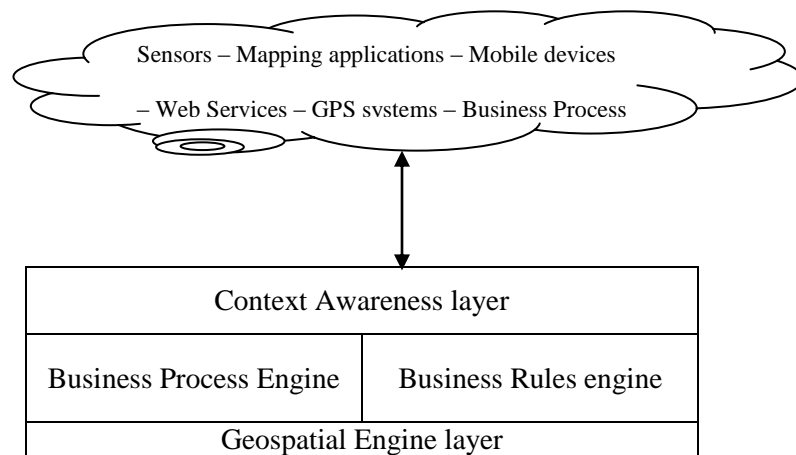


Figure 17: Geospatial Context Awareness Business process overview

As shown in Figure 17, the context layer communicates with the environment or context of the application and because of the layered approach, the information flows from the

context layer to the business process, business rules, and geospatial layers. In the following sections, we present the details of each layer.

6.1 Geospatial Engine layer

The Geospatial layer implements geographic operations, projections, geospatial relations and any other geospatial utilities that it is needed by the upper layers. The layer is divided into two sub-layers, as shown in Figure 18.

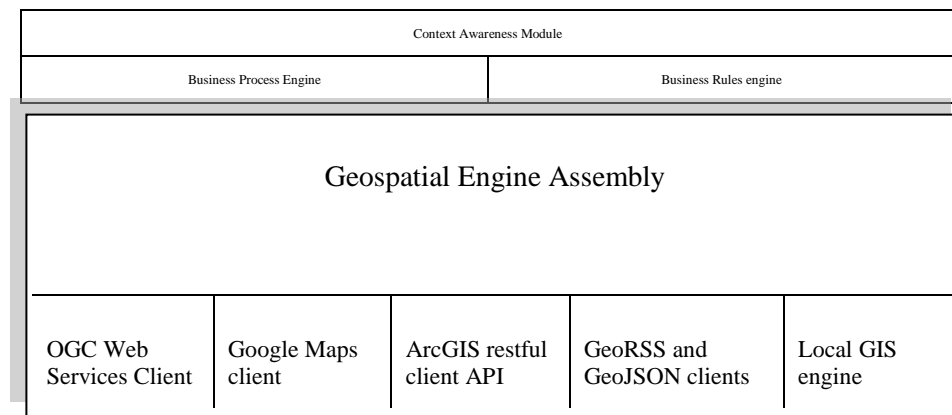


Figure 18: Geospatial Engine layer

The geospatial layer implements a series of Web Service clients for the different geographic Web Services standards. The most common standards used today are the OGC compliance services, such as web features services (WFS), web map services (WMS), web coverage service (WCS); Google Maps RESTful API; ArcGIS RESTful API; GeoRSS; and GeoJSON services. The Geospatial Engine acts as a middleware that connects these Web Services and local GIS engines to the remaining layers. Local GIS engines are usually defined as APIs in different programming languages, such as C, Python, and Java, which implement geometry operations in an efficient way. The Geospatial Engine assembly sub-layer acts as a catalog and an invocation mechanism for

all of the Web Services and local engines, as well as aggregating the results from the Web Services into artifacts that can be used in the business process and business rules. This sub-layer implements different parsers, such as XML, GML, JSON, and any other response types from the Web Services and local engines. After the parsing process, the results are assembled into meaningful artifacts for the business process and business rules, such as GeoFacts, Incidents, and input datasets for the business process. In summary, the Geospatial Engine is a library that encapsulates the operations and model of the geographic domain.

6.2 Business Process and Business Rules Layer

The second layer has the business process engine and the business rules engine. The business rules engine has two sub-layers. The first sub-layer is the basic rules engine implementing the RETE algorithm. The second sub-layer has the implementation for the geospatial rules using our domain specific language called Geospatial-DL or GDL. Figure 19 shows both sub-layers.

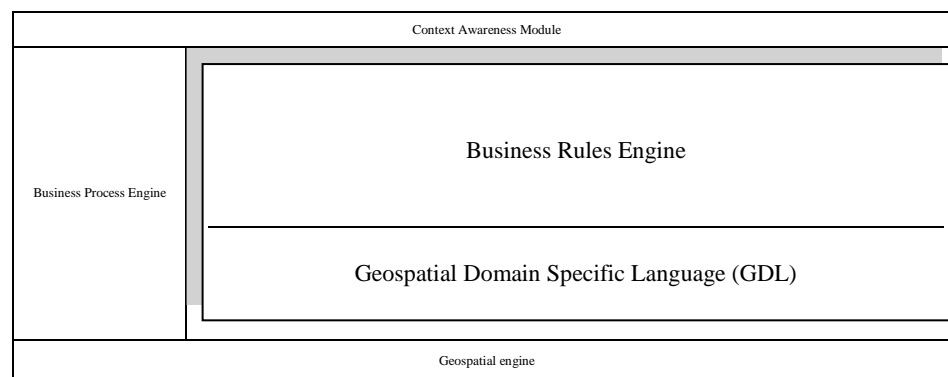


Figure 19: Business Rules Engine with Geospatial reasoning support

Defining the extension in GDL allows the engine to be independent from the implementation of the geospatial reasoning module. GDL enables standardization for the engine and then uses a parser to convert the rules into functions to determine the relationship between facts. For example, in Figure 20, the proximity closeby rule is defined in GDL and then is parsed for the business rules engine to execute it. As seen in the example, the business rule will first relate all the different GeoFacts in the working memory. This is achieved using a global identifier for the GeoFact, which is generated by the engine when the fact is registered in the knowledge base. Then, using the geospatial engine API, the distance is calculated and compared with the variable pass in the GDL statement.

GDL	
rule	
when	GeoFact1(this closeby [5km] GeoFact2)
then	<i>Action...</i>
Rule after parsing	
rule	
when	g1:GeoFact() g2:GeoFact(g1.id != this.id) eval(GeoSpatialEngine.distance(g1.geometry, g2.geometry)<5km)
then	<i>Action...</i>

Figure 20: GDL to Production Rule

The business process sub-layer is divided into three different modules: a business process virtual machine, a constraints validator module, and a geospatial extension to support geographic constraints, as seen in Figure 21. The business process virtual executes valid BPMN documents. This engine can be a native BPMN execution or a BPEL engine. The

virtual machine is in constant communication with the constraints validator module. When the business process is about to execute a task and GCL constraints have been specified for that task, the business process engine calls the constraints validator. The GCL engine must have a record of all the process instances running in the business process engine so it can apply the constraint to the correct instance of the process. The constraints validator returns a Boolean value, which represents the validation of the constraint and the business process, taking into account that the return value, must adapt and react to the constraint. The geospatial constraint interface implements the series of patterns described in Chapter 4 and therefore extends the constraints validator to support geospatial constraints. This interface uses the geospatial API from the GeoSpatial Engine to implement geospatial operations into the business process. This design allows the system to be modular and extensible.

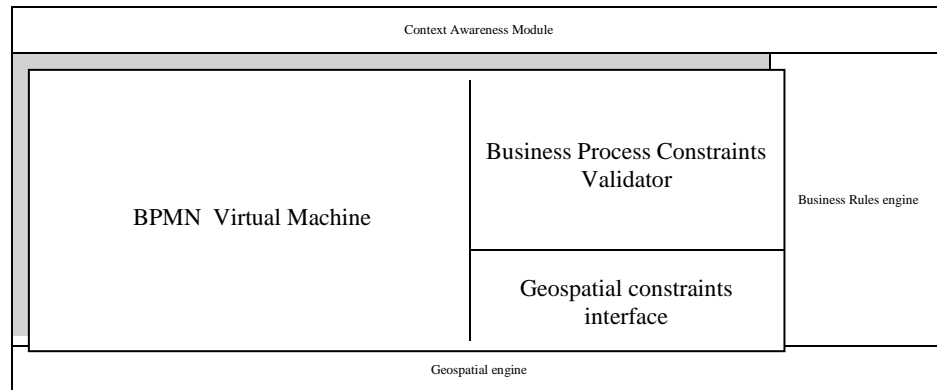


Figure 21: Business Process Engine with Geospatial constraints support

6.3 Context Awareness Layer

The upper layer is the context awareness module. This layer is divided into four sub-layers, a geospatial context query module, a geospatial events handler and publisher,

a geospatial context services, and a communication interface, as seen in Figure 22. This last sub-layer, the communication interface, controls the communication between the business process and other applications or systems. The communication interface takes input from differences sources, such as Web Services, sensors, GPS, mobiles, and others. The information passes through the others three sub-layers to the business process and business rules engines. For example, when an event from a sensor is registered by the context of the system, the event is also registered in the events handler with a callback handler for the specific event. A handler is an action that activates part of the process instance through events. Also, the event is registered in the knowledge base where it will become a GeoFact or GeoEvent for the business rules engine. If the business process needs to query the context, then it will do so through the context query sub-layer, which uses the communication interface to query the context for the information that the business process requires. If the business process requires information from a Web Service which is registered in the context service sub-layer, then the communication interface invokes the service and registers the result in the events handler layer. The services registered in this sub-layer provide contextual information about the environment of the business process. In order for the process to communicate its status or events to the context, the business process registers events in the event publisher layer. The communication interface then parses these events and publishes the events of the process in a standard way such as Atom or RSS.

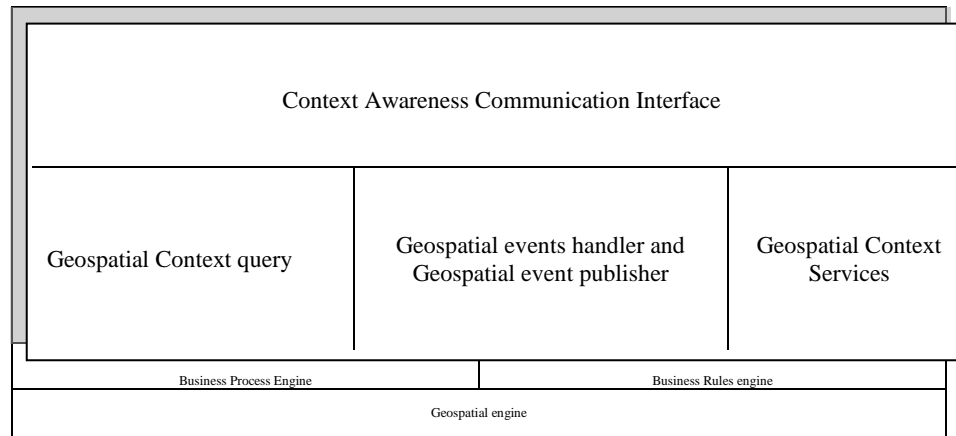


Figure 22: Context awareness layer

CHAPTER 7

GEOFLOW: PROTOTYPE IMPLEMENTATION AND EVALUATION

7.1 Implementation details

The prototype implementation was developed in Java and Python. It also relies on many Web Services from ArcGIS online [17] and Google Maps [16]. The prototype validates the concepts presented in the previous chapters. Figure 23 shows an overview of the system and the different frameworks used.

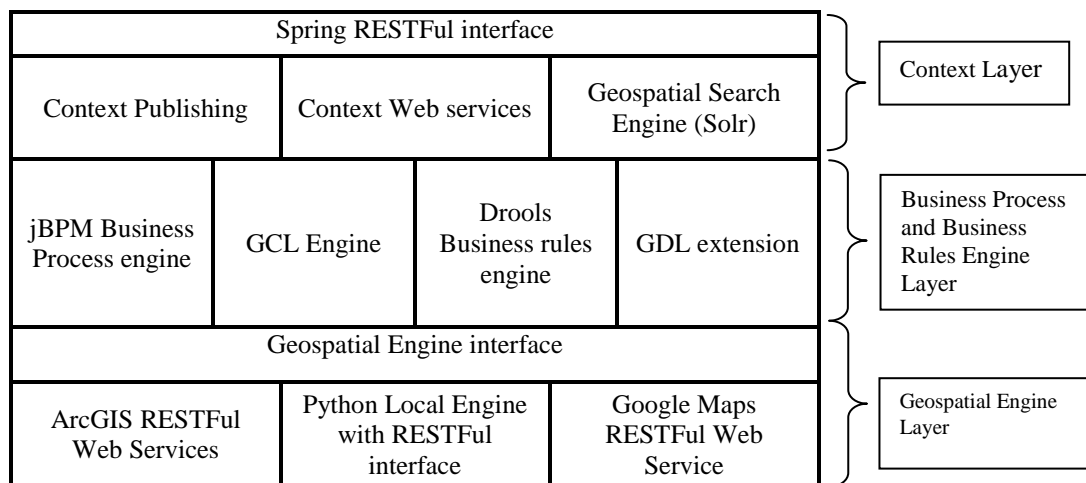


Figure 23: Overview of the prototype system

Geospatial Engine

The Geospatial layer is divided into two sub-layers: 1) GIS Web Service client sub-layer and 2) the geospatial interface sub-layer. The first layer implements different

Web service calls to services in ArcGIS online, Google Maps, and a local Python engine with a RESTful interface. The local engine was implemented using the Python shapely [34] library which has an API to support geometric operations, the Python pyproj [35] library which has an API to perform geographic projections, and the Python webpy [36] library which is a framework to develop CGI scripts. The geospatial operations implemented using the Python engine are distance, buffer, within, cross, disjoint, contains, intercepts, touch, and measure. All the operations have two parameters in GeoJSON format and return a JSON response via HTTP GET or HTTP POST. The Python engine supports input as latitude and longitude and projects it to Web Mercator. ArcGIS services are used mainly for querying ArcGIS RESTful data services and Google Maps services are used to calculate driving distance and driving time. The second sub-layer of the Geospatial engine converts the output of a Web Service call into meaningful artifacts for the business process and the business rules engines. Figure 24 shows the UML diagram of the classes used by the Geospatial engine.

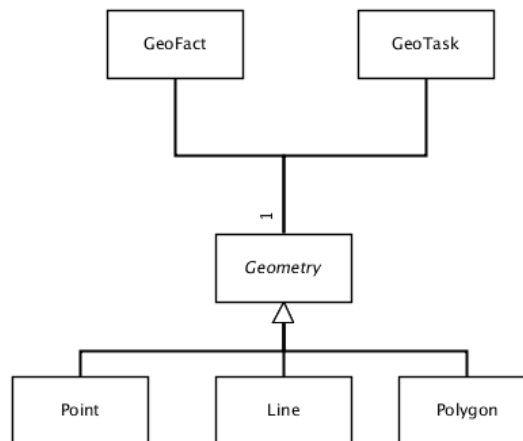


Figure 24: UML classes diagram for Geospatial Engine

The Geospatial Engine is a static class which exposes an API to invoke the Web Services in a seamless way. The parameters of most of the methods are of type Geometry which is an abstract class. Each child of the Geometry class implements methods to support conversion between GeoJSON and geometry objects. As seen in the class diagram, GeoFact and GeoTask have a geometry object which then will be used as parameter to invoke the Geospatial Engine.

Business Process Engine and GCL Engine

The business process engine and modeler used for the prototype was the open source version of jBPM 5.2 [37] which is a product from JBoss Redhat. The business process model was implemented using jBPM eclipse plugin and the execution engine is the jBPM engine for BPMN 2.0. Even though jBPM does not support at this version the full specification of BPMN 2.0, it provides access to the process engine API as well as an easy way to extend work items or tasks making it ideal for the implementation. The GCL Engine was implemented to extend jBPM in order to support GCL constraints. For each process instance, a GCL Validator instance is created. The GCL engine keeps track of the GeoTasks that are being executed in the process instance.

Figure 25 shows the steps of the GCL Engine and the process instance interactions. When a process instance is created, a new GCL Validator is created by the GCL Engine and is associated with the process instance.

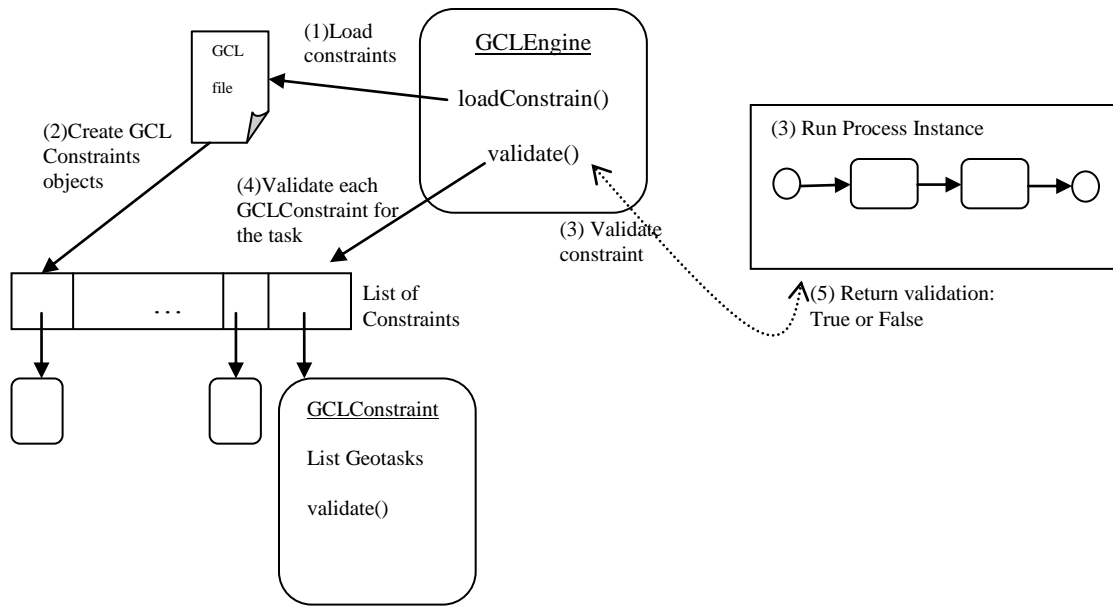


Figure 25: Process instance and GCL Engine interaction

The GCL Engine performs the following steps when a process instance requires a validator:

- 1) The GCL Engine loads the constraints from a GCL file stored in the file system. This file contains the constraints for the process in JSON format.
- 2) The GCL engine creates a GCL Constraint object for each constraint in the JSON document. Each constraint object has the associated GeoTasks, the relationship or constraint definition, and the parameters for the operation, if needed. Each GCL Constraint object has a validate method which returns valid, if the constraint is met, invalid, if not met, or nocheck, if the constraint cannot be validated because the reachability property, which states that the constraint is validated when all the task of the constraint are reached in the process execution.

- 3) When a GeoTask is reached the process instance calls the GCL Engine through the GCL Validator assigned to the process.
- 4) The engine has a method named `validate` which goes through all the GCL constraint for the current task and perform the validation.
- 5) The validation returns *true* if all the constraints return *valid* or *nocheck*, and it returns *false* when at least one constraint is false.

Business Rule Engine

The prototype uses the open source version of Drools 5.3 [38], which it is also in the same product family of jBPM from JBoss Redhat. Drools provides a domain specific language syntax specification to create extensions to the framework. Drools also supports complex event reasoning and easy integration with business processes. The rules engine was extended by creating a new artifact called GeoFact. This class has a reference to a geometry object and has a unique id when inserting it to the knowledge base. Each geospatial rule was defined using GDL. Using a parser provided by Drools, the rule is transformed to perform geospatial validations using the Geospatial Engine API and the geometry reference of each GeoFact. With this extension, the rules engine supports geospatial reasoning along with any other type of reasoning supported by Drools. In section 7.2, a full example is shown that explains the details of the language.

Context Layer

The context layer has four main components, which are the Spring framework [39] interface, the context publishing module, the well-known context module, and the

geospatial search engine. The Spring framework provides a RESTful interface that allows the GEOFlow system to interact with the environment, and vice versa. Through this interface an application communicates with the business process and registers events to the business rules engine. The geospatial search engine provides one of the three components of the business context. This module was implemented using Solr [40], which is project from Apache Software Foundation. Solr provides a full text search engine and geospatial queries based on distance. The geospatial Web Services module serves as a catalog for the business process. The different Web Services that the business process needs are registered in the catalog and exposed as API client calls for the business process. The context publishing module implements the way that the process registers events with the context. The module is an RSS writer that provides information of the events that the process registers.

Map viewer helper tool

An application using Google Maps [16] and ExtJS [41] was developed to interact with the prototype system. This application communicates with the GEOFlow system through the context layer, specifically, the Spring framework interface. The application shows in a map the different use cases developed to illustrate the main points of this thesis. The application is not part of the system, and only interacts and display data in the map. Figure 26 shows an overview of the Web application.

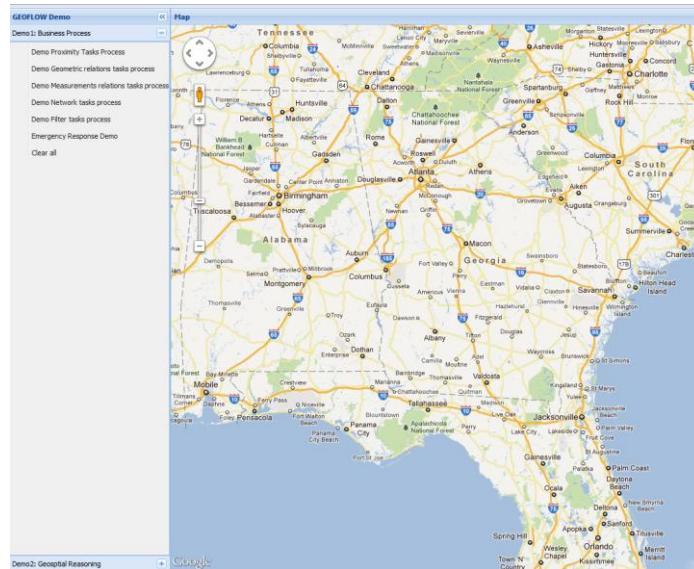


Figure 26: Web Application used to interact with the GEOFlow system

7.2 Emergency Response System Use Case and Evaluation

In the next section, an emergency response system use case is presented validating the concepts presented in chapter 5. This implementation illustrates the use of the proposed concepts in a real life scenario. Appendix A describes a generic implementation for each GCL constraint group. Appendix B describes a generic implementation for each GDL geospatial business rules.

GCL and Emergency Response system BPMN

The following example shows an emergency response system with geospatial constraints. Figure 27 shows the situation of the emergency in a Map. Figure 28 shows the actual business process to respond to the emergency. The description of the problem is as follows:

- The input of the workflow is a tornado path, which is represented as Line geometry.
- The workflow must validate if the emergency is supported by the process. This validation takes place in the first task of the process. The workflow must accept only tornado paths that are less than 35km in length. If the tornado is greater in length than 35km then the workflow must signal FEMA, so they can respond to the emergency and then the process terminates. If the process is supported by the workflow then it continues and performs an assessment.
- There are three tasks in parallel in the workflow. They are hospital preparedness, schools notification, and activate shelters.
- The hospital preparedness calls a Web Service to get the hospitals in the area. Then, it validates whether the hospitals are near by at least 7km of the tornado path. If the hospital meets the constraint then the process activates the task to prepare the hospital for the emergency.
- School notification calls a Web Service to get the schools in the area. Then it validates whether the schools are within a tornado buffer of 5km. If the school meets the constraint then the process notifies the school to take action over the emergency.
- Activate shelters calls a Web Service to get the shelters in the area. Then, it validates whether the shelter is north of the tornado (which is the projection of the tornado path) and whether the shelter is distant by at least 9km. If the shelter meets the constraints, then the process activates the shelter to be open to receive people from the community.

- The last task validates whether the tornado is over a community. The community is represented as a polygon constant in the constraint. If the tornado is over the community then this community gets an alert.

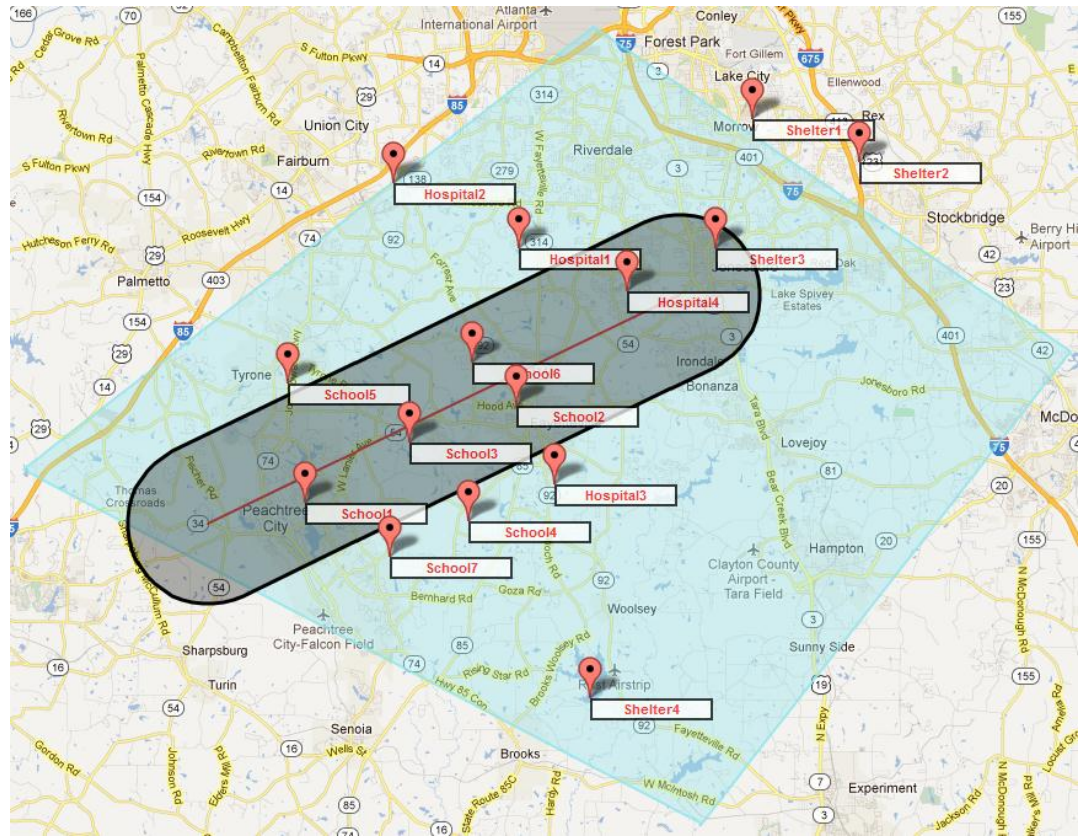


Figure 27: Emergency situation representation in the map

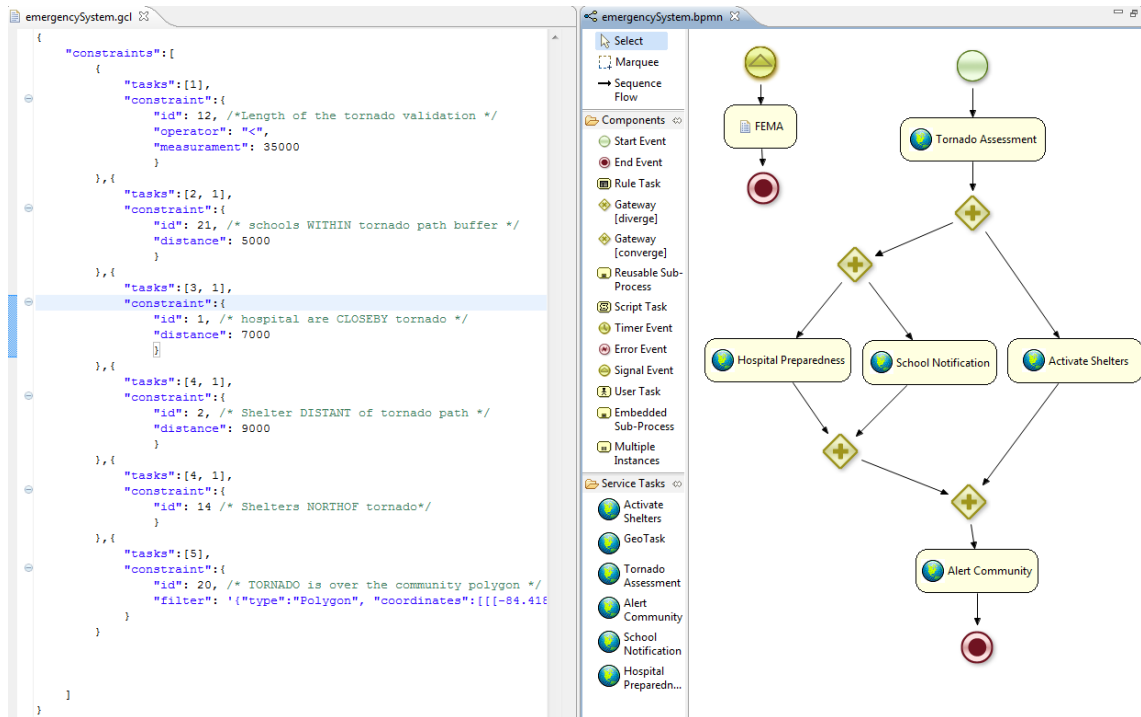


Figure 28: Emergency response system in BPMN with GCL constraints

The first task of the process execution is the Tornado Assessment as seen in the output of Figure 29. As seen in Figure 28, the task performs other validations which are omitted due the reachability property. The second task executed is the schools notification. The task is a service task which calls a School Web Service. The constraint is applied to each school of the Web service's respond. As seen in Figure 29, the schools 1, 2, 3, and 6 are notified.

```

Console
<terminated> EmergencyResponseTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Mar 1, 2012 10:16:57 AM)
-----
Tornado Assessment
-----
Checking whether the process support the tornado based on the length
Tornado length = 32614.121
Validation TRUE = GCLConstraint[ name: LENGTH, tasks:[1 ], params: [< 35000 ]]
Validation NOCHECK(reachability property) = GCLConstraint[ name: WITHINBUFFER, tasks:[2 1 ], params: [5000 ]]
Validation NOCHECK(reachability property) = GCLConstraint[ name: CLOSEBY, tasks:[3 1 ], params: [7000 ]]
Validation NOCHECK(reachability property) = GCLConstraint[ name: DISTANT, tasks:[4 1 ], params: [9000 ]]
Validation NOCHECK(reachability property) = GCLConstraint[ name: NORTHOF, tasks:[4 1 ], params: []]
Tornado supported, continue...
-----

Schools Notification
-----
Simulating call WS to get schools, then validate
+++++
Validation TRUE = GCLConstraint[ name: WITHINBUFFER, tasks:[2 1 ], params: [5000 ]]
Validation Constraints = TRUE
[School 1] = notified
+++++
Validation TRUE = GCLConstraint[ name: WITHINBUFFER, tasks:[2 1 ], params: [5000 ]]
Validation Constraints = TRUE
[School 2] = notified
+++++
Validation TRUE = GCLConstraint[ name: WITHINBUFFER, tasks:[2 1 ], params: [5000 ]]
Validation Constraints = TRUE
[School 3] = notified
+++++
Validation FALSE = GCLConstraint[ name: WITHINBUFFER, tasks:[2 1 ], params: [5000 ]]
Validation Constraints = FALSE
[School 4] is not within tornado path
+++++
Validation FALSE = GCLConstraint[ name: WITHINBUFFER, tasks:[2 1 ], params: [5000 ]]
Validation Constraints = FALSE
[School 5] is not within tornado path
+++++
Validation TRUE = GCLConstraint[ name: WITHINBUFFER, tasks:[2 1 ], params: [5000 ]]
Validation Constraints = TRUE
[School 6] = notified
+++++
Validation FALSE = GCLConstraint[ name: WITHINBUFFER, tasks:[2 1 ], params: [5000 ]]
Validation Constraints = FALSE
[School 7] is not within tornado path
-----

```

Figure 29: Output of Tornado Assessment and School Notification task in the Emergency response system BPMN

The next task to be executed is the Hospital preparedness GeoTask. This task validates whether the hospitals are nearby 7km of the tornado path. If the hospital is close by the tornado then the task performs the necessities steps to prepare the hospital for the emergency. As seen in Figure 29, the hospitals that need to be prepared are the hospital 2 and hospital 4.

```

-----
Hospital preparedness
-----
Simulating call WS to get Hospitals, then validate
+++++
Validation TRUE = GCLConstraint[ name: CLOSEBY, tasks:[3 1 ], params: [7000 ]]
Validation Constraints = TRUE
[Hospital 1] = prepared
+++++
Validation FALSE = GCLConstraint[ name: CLOSEBY, tasks:[3 1 ], params: [7000 ]]
Validation Constraints = FALSE
[Hospital 2] is not close to tornado, then stand by
+++++
Validation FALSE = GCLConstraint[ name: CLOSEBY, tasks:[3 1 ], params: [7000 ]]
Validation Constraints = FALSE
[Hospital 3] is not close to tornado, then stand by
+++++
Validation TRUE = GCLConstraint[ name: CLOSEBY, tasks:[3 1 ], params: [7000 ]]
Validation Constraints = TRUE
[Hospital 4] = prepared
-----

-----
Activate shelters
-----
Simulating call WS to get Shelters, then validate
+++++
Validation TRUE = GCLConstraint[ name: DISTANT, tasks:[4 1 ], params: [9000 ]]
Validation TRUE = GCLConstraint[ name: NORTHOF, tasks:[4 1 ], params: []]
Validation Constraints = TRUE
[Shelter 1] activated
+++++
Validation TRUE = GCLConstraint[ name: DISTANT, tasks:[4 1 ], params: [9000 ]]
Validation TRUE = GCLConstraint[ name: NORTHOF, tasks:[4 1 ], params: []]
Validation Constraints = TRUE
[Shelter 2] activated
+++++
Validation FALSE = GCLConstraint[ name: DISTANT, tasks:[4 1 ], params: [9000 ]]
Validation TRUE = GCLConstraint[ name: NORTHOF, tasks:[4 1 ], params: []]
Validation Constraints = FALSE
[Shelter 3] does not meet criteria to activate
+++++
Validation TRUE = GCLConstraint[ name: DISTANT, tasks:[4 1 ], params: [9000 ]]
Validation FALSE = GCLConstraint[ name: NORTHOF, tasks:[4 1 ], params: []]
Validation Constraints = FALSE
[Shelter 4] does not meet criteria to activate
-----

```

Figure 30: Output of Hospital preparedness task and the Activate shelter task in the
Emergency response system BPMN

The next task to be executed is the Activate Shelters GeoTask. This task combines two constraints. The first constraint is whether the shelter is north of the tornado path. The second constraint is whether the shelter is distant by 9km of the tornado. The shelters that

need to be activated are shelter 1 and shelter 2. This task shows how to use more than one constraint in one task. The last task is the Alert community GeoTask. The constraint validates whether the tornado is over the polygon boundaries of the community. The task alerts the community if the constraint returns true. As one can see in the example, the process was executed validating every constraint. The decision of the actions taken when the validation returns either true, false, or no check are leave to process designer. In this case, the validation serves to make a decision about the process execution, to filter output datasets, and to take direct action over a task execution.

```

-----
Alert community -- precondition
-----
Validation TRUE = GCLConstraint[ name: FILTER, tasks:[5 ], params: [{"type":"Polygon","coo
Validation Constraints = TRUE
Alerting community of potential tornados
-----

```

Figure 31: Output of Alert community task in the Emergency response system BPMN

Emergency Response Business Rules:

The following example shows an emergency system response business rules. The rules implemented in this use case are the following:

- The GeoFacts inserted into the knowledge-base are the tornado path (Line), the tornado's center (Point), Atlanta location (Point), the schools in the area (Point), and the hospitals in the area (Point). The id for each GeoFact represents the type of the GeoFact.
- Evacuation of Schools within the tornado path rule: This rule filter of the schools that are within 5km of a buffer of the tornado path. If a GeoFact is detected, the system triggers the process to evacuate the schools.

- Hospital response rule: This rule filters the hospitals that are between 6km and 8km of the tornado path. These hospitals are designed to respond to the emergency.
- Declare state of emergency in south of Atlanta rule: This rule verifies if the tornado's center is south of Atlanta and declares the emergency in this part of the city.
- Hospital for each school assignment: This is a complex rule that assigns the closest hospital to each school. This rule calculates the distance for each school and each hospital. Then it finds from the hospitals that are assigned to respond to the emergency the closest one and assign it to the school.

Figure 32 shows the situation in the Map and the GeoFacts that are inserted in the knowledge base. Figure 33 and Figure 34 show the Emergency response rules.

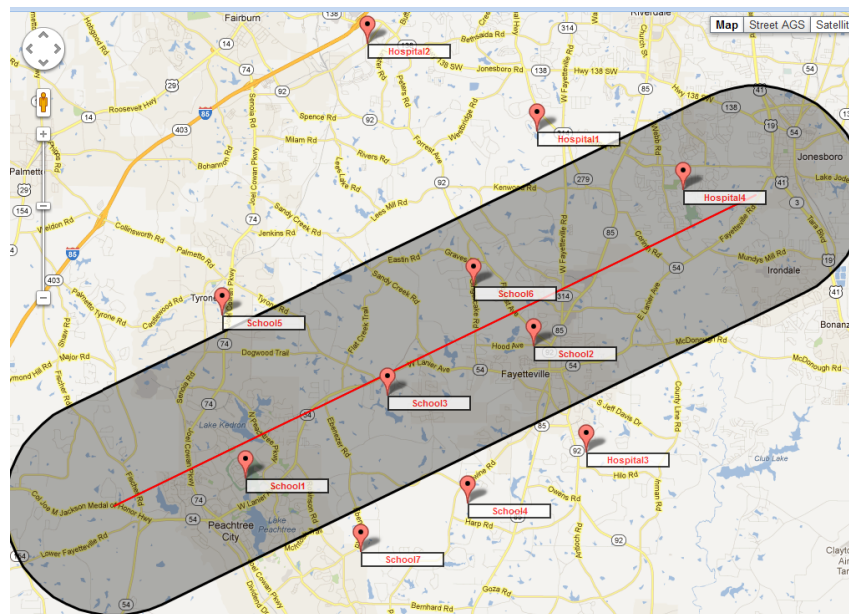


Figure 32: Tornado in the south of Atlanta


```

package edu.uga.research.rules

#list any import classes here.
import edu.uga.research.util.model.*;
import edu.uga.research.util.engines.GeoSpatialEngine;

rule "Evacuation of School within tornado path"
when
    $g1:GeoFact(id == "tornado")
    $g2:GeoFact(id == "school")
    eval( GeoSpatialEngine.inBuffer($g2.getGeometry(), $g1.getGeometry(), 5000 ) )
then
    System.out.println("Evacuate School --> "+$g2.getDescription());
end

rule "Hospital to respond emergency"
when
    $g1:GeoFact(id == "tornado")
    $g2:GeoFact(id == "hospital")
    eval( GeoSpatialEngine.distance($g2.getGeometry(),
        $g1.getGeometry() )>6000 && GeoSpatialEngine.distance($g2.getGeometry(),
        $g1.getGeometry() )<8000 )
then
    System.out.println("Hospital Selected to respond emergency --> "+$g2.getDescription());
end

rule "Declare State Emergency in South ATL"
when
    $g1:GeoFact(id == "centerTornado")
    $g2:GeoFact(id == "atlanta")
    eval( GeoSpatialEngine.isSouthOf($g1.getGeometry(), $g2.getGeometry() ) )
then
    System.out.println("Tornado is South ATL, Declare Emergency in South ATL ");
end

```

Figure 33: First set of rules for tornado emergency

```

rule "Hospital for each School assignment"
when
    GeoFact(id == "evacuation")
    $g1:GeoFact(id == "tornado")
    $g2:GeoFact(id == "school")
    eval( GeoSpatialEngine.inBuffer($g2.getGeometry(), $g1.getGeometry(), 5000 ) )
    $g3:GeoFact(id == "hospital")
    eval( GeoSpatialEngine.distance($g3.getGeometry(), $g1.getGeometry() )>6000
        && GeoSpatialEngine.distance($g3.getGeometry(), $g1.getGeometry() )<8000 )

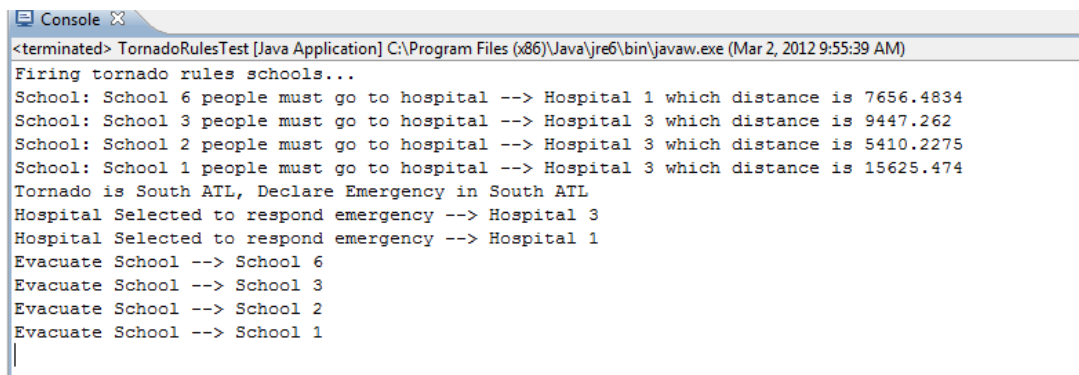
    accumulate(
        $g4:GeoFact(id == "hospital") and
        eval( GeoSpatialEngine.distance($g4.getGeometry(), $g1.getGeometry() )>6000
            && GeoSpatialEngine.distance($g4.getGeometry(), $g1.getGeometry() )<8000 ),
        $min: min( GeoSpatialEngine.distance($g4.getGeometry(), $g2.getGeometry() ) )
    )

    eval( GeoSpatialEngine.distance($g3.getGeometry(), $g2.getGeometry() ) == $min.doubleValue() )
then
    System.out.println("School: "+$g2.getDescription()+" people must go to hospital --> "+$g3.getDe
end

```

Figure 34: Complex rule for tornado emergency. Assigns hospital to each school based on proximity

Figure 35 shows the output of the emergency response rules reasoning process. The engine has found that Schools 1, 2, 3, and 6 needs to be evacuated. Hospital 1 and 3 are suitable for the response. Even though hospital 4 is closest to the tornado path, the engine knows that the hospital is within the tornado path and it may be not suitable to respond to the emergency. The tornado is south of Atlanta and the engine declares the state of emergency for south of Atlanta. The schools are assigned to each hospital suitable to respond and closest to the school. School 6 gets assigned to hospital 1, School 1, 2, and 3 get assigned to hospital 3.



```
<terminated> TornadoRulesTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Mar 2, 2012 9:55:39 AM)
Firing tornado rules schools...
School: School 6 people must go to hospital --> Hospital 1 which distance is 7656.4834
School: School 3 people must go to hospital --> Hospital 3 which distance is 9447.262
School: School 2 people must go to hospital --> Hospital 3 which distance is 5410.2275
School: School 1 people must go to hospital --> Hospital 3 which distance is 15625.474
Tornado is South ATL, Declare Emergency in South ATL
Hospital Selected to respond emergency --> Hospital 3
Hospital Selected to respond emergency --> Hospital 1
Evacuate School --> School 6
Evacuate School --> School 3
Evacuate School --> School 2
Evacuate School --> School 1
```

Figure 35: Output after firing the Emergency response rules

Business Process Geospatial Context

The business process geospatial context module was implemented by creating a Process Context class that serves as interface for the business process and business rules engines to interact with the external context. The first component is a geospatial search client. This client communicates through a RESTful API with the Solr search engine. The context provides a generic search method that receives as parameters a text query, point location, and distance. The geospatial search engine matches the documents that are nearby by a factor of the distance parameter and the text query. The Web Services catalog

module was implemented using different Web Services from ArcGIS Server and Google Maps. This implementation is a simple Web Service client that invokes the services and then parsers the results. The publishing events module was implemented using Rome RSS writer API. The Process Context object records the events that tasks within the process triggered during the process execution. Then through the Spring Framework interface the RSS events feed is exposed to other applications or users. The RSS feed contains basic properties of the events as well as location information. Finally, the external events detection module or register events module was implemented using Drools with events detection support. The application, users, or other systems registers events with location information through the Spring framework interface. This interface then registers the GeoEvents into the knowledge base and the business rules engine matches the events using the rules specified by the system.

Figure 36 shows a BPMN process with context support. This process has a reference to the Process Context Engine. Each task can invoke the context depending on the needs of each implementation. Figure 37 shows the GeoEvents register in the context. The application uses the RESTful interface to send the point location and name when the user clicks within the map.

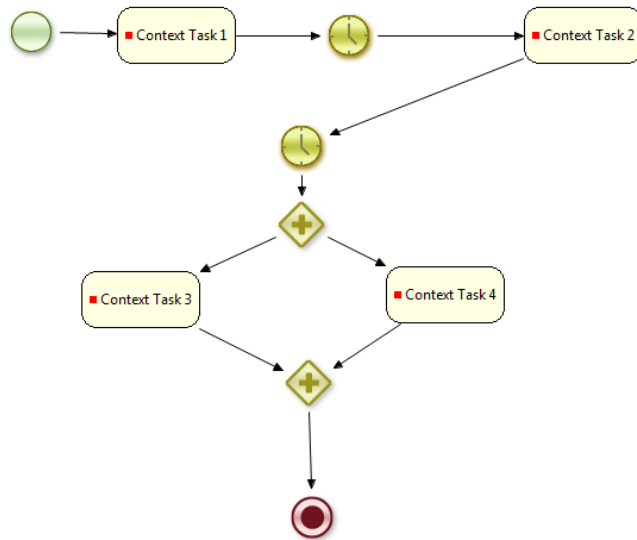


Figure 36: Business Process with Geospatial context awareness

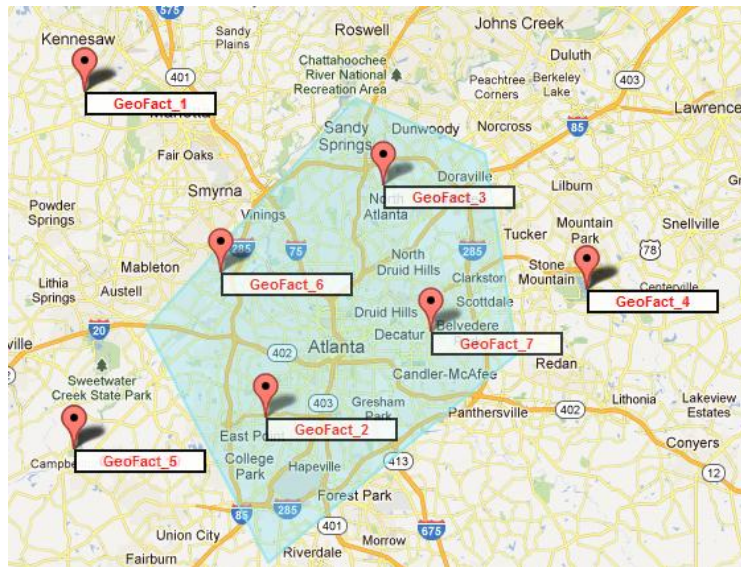


Figure 37: GeoEvents register to the GEOFlow system through the Map application

Figure 38 shows the business rules used to parse the events. The rules match GeoEvents that within the Atlanta area. If the GeoEvents at certain point are more than three, then the rules engine start the BPMN process.

```
package edu.uga.research.context.bpmn

#list any import classes here.
import edu.uga.research.util.model.*;
import edu.uga.research.util.engines.GeoSpatialEngine;
import java.util.*;
import edu.uga.research.context.*;

global edu.uga.research.context.ProcessContext context;

declare GeoFact
@role( event )
end

rule "Events in the Atlanta Area"
when
    $g1:GeoFact(id=="atlanta")
    $g2:GeoFact(this!=$g1, description != "warning")
    eval( GeoSpatialEngine.isWithin($g1.getGeometry(), $g2.getGeometry() ) )
then
    $g2.setDescription("warning");
    update($g2);
    System.out.println($g2+" is the Atlanta Area set to warning ");
end

rule "Warning events > 3 then activate process"
when
    Number($n : intValue) from accumulate( $e2: GeoFact(description == "warning" ), count($e2)
    eval($n > 3 )
then
    System.out.println("More than 4 events has been detected as warning => Start BPMN process");
    #Start the process
    Map<String, Object> params = new HashMap<String, Object>();
    params.put("context", context);
    drools.getKnowledgeRuntime().startProcess( "bpmn_using_context", params);
end
```

Figure 38: Business rules used to filter GeoEvents

As seen in Figure 39, the first task uses the context object to invoke a Web Service (temperature in a given point) part of the context. The task 2 and task 3 perform another context invocation but this time getting the hospitals in the state of GA and Counties that intercepts with point geometry respectively. Task 4 invokes the Solr search engine with a text query “fire stations” and a point location with a distance of 16km. As seen in the output, the engine returns the nearby stations, as well as other features that matched the

query. Tasks 1, 2, and 3 register events to the context publisher. Figure 40 shows the RSS feed of the two processes execution.

```
GeoFact[id=GeoFact_2, geometry=Point] is the Atlanta Area set to warning
GeoFact[id=GeoFact_3, geometry=Point] is the Atlanta Area set to warning
GeoFact[id=GeoFact_6, geometry=Point] is the Atlanta Area set to warning
GeoFact[id=GeoFact_7, geometry=Point] is the Atlanta Area set to warning
More than 4 events has been detected as warning => Start BPMN process

-----

Task 1 - Querying Temperature in Point(33.92,-84.36) =12.0C

-----

Task 2 - Querying ESRI hospitals in GA. count =218

-----

Task 3 - Spatial Query Point(33.87744,-83.3612399979999) intercepts ESRI COUNTIES in GA
attributes: {[CNTYIDFP = 13219] [SHAPE.area = 0.047016274] [FUNCSTAT = A] [NAME = Oconee] [STATEFP = 13] [METDIVFP = ] [CBSAFP = 
attributes: {[CNTYIDFP = 13059] [SHAPE.area = 0.030574545] [FUNCSTAT = C] [NAME = Clarke] [STATEFP = 13] [METDIVFP = ] [CBSAFP = 

-----

Task 4 - Doing a FULL TEXT SEARCH WITH GEOSPATIAL SUPPORT OVER MORE 150K documents
Task 4 - Query Solr( 'fire stations',32.3418,-84.99232, 16km)
firestation - firestation_25
firestation - firestation_29
historic - historic_48023
historic - historic_80063
historic - historic_48187
historic - historic_48049
arch - arch_8877
arch - arch_16642
arch - arch_20411
arch - arch_25679
arch - arch_33366
arch - arch_6481
arch - arch_14282
arch - arch_19968
arch - arch_22023
arch - arch_33154

-----
```

Figure 39: Business Process with context awareness output

GEOFlow Events publishing

GEOFlow Process register events to the environment in RSS format

[Event on task=1](#)

Thursday, March 08, 2012 9:02 AM

Task 1 query the temperature, processId =1

[Event on task=2](#)

Thursday, March 08, 2012 9:02 AM

Task 2 has the count of hospitals in GA, processId =1

[Event on task=3](#)

Thursday, March 08, 2012 9:02 AM

Task 3 knows the counties in point(33.87744,-83.3612399979999), processId =1

[Event on task=1](#)

Thursday, March 08, 2012 9:04 AM

Task 1 query the temperature, processId =2

[Event on task=2](#)

Thursday, March 08, 2012 9:04 AM

Task 2 has the count of hospitals in GA, processId =2

[Event on task=3](#)

Thursday, March 08, 2012 9:04 AM

Task 3 knows the counties in point(33.87744,-83.3612399979999), processId =2

Figure 40: GEOFlow RSS feed for business process events

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

This thesis shows the importance of integrating geospatial information into business processes. The approach uses the basics of geospatial analysis to fully integrate with the major components of a business process, which are the model, the inference module, and the context of the workflow. The implementation of the project shows the viability of the concepts explained in the previous chapters. The next generation of business processes is the ability to fully describe real scenarios into workflow systems. One major component is the geospatial data and operations. Our contributions make it easier for a business process to achieve the business goal for which it was created.

Constraints and non-functional requirements are an important part of a business process model. In this thesis, geospatial information and operations are integrated seamlessly into the business processes without altering its semantics, but rather enriching it. These constraints and requirements can be represented as preconditions, postconditions and invariants that validate the execution of the task. This was achieved by proving a geospatial constraint language (GCL) which describes the constraints within the task level in a declarative way. GCL represents the basic constructors of geospatial operations and enforces the process instance to validate the constraint in execution time. A proposal

to express these constraints in the XML document of BPMN model, as well as a graphical representation for some of the patterns, have been presented.

The second part of the thesis extended the business rules engine to support geospatial reasoning. Geospatial reasoning provides full business intelligence using inference through production rules. The introduction of GeoFact and GeoEvent allows the engine to analyze information in real time based on the knowledge base and to produce desired actions. This was achieved by using a formal extension expressing geospatial relationships between facts with a Geospatial Domain Language (GDL). These geo-rules can be easily integrated into the business rules engine and can enable complex event detection based on geographic location. The final part of the thesis described the geospatial requirements to implement the interaction between the business process and the geospatial context. This was achieved by extending the context framework to support geospatial queries, geo event handling and publishing, and geospatial services. The context based on geospatial information allows the business process to be able to adapt, depending on the geographic environment which makes the process more flexible and aware of the surrounded context. A layered approach was presented that shows the implementation of the previous concepts into a business process system. The layered approach and the modularization of the geospatial components allow the engine to integrate the geospatial extensions without major deployments or modifications of current business process systems. These three components can be used in many applications. This thesis modeled an emergency response system example using the proposed system which serves as concept validation and implementation.

8.2 Future work

This thesis has proposed to express constraints, requirements, reasoning, and context definition into business process systems. The study's approach was focused in the geographic domain and its implications of implementation and design of the system. From the business process model perspective, there is a need to define constraints and non-functional requirements in a standard way. Business processes need a language similar to OCL to define process constraints and also an easy way to incorporate different domains constraints into business processes through a standard language. BPMN needs to be extended to represent real life processes with constraints in the task and workflow level.

Geospatial rules provide the inference engine with a way to process facts according to geographic location and relationships in a knowledge database. Geospatial reasoning can be used alongside temporal reasoning to support geo-temporal validations in the inference engine. This integration will enable enterprise systems to process complex events based on timestamps and location, which opens more possibilities to model real life rules and actions. The context of a business process needs to be standardized in a way that the model can interact with a common interface without depending on the application domain. Geospatial context opens the door to explore a formal definition of the context of a business process in a more generic way. The context definition can be achieved by using a generic ontology that supports extensions to domain specific ontologies where the business process can interact by API calls that retrieve the information needed to adapt to the current state of the environment.

From the implementation perspective, a formal constraint engine needs to be defined in a way that is fully integrated with the business process engine. The constraints of a process should be integrated within the business process model in way that is flexible and evident to the business analyst designer to understand and change the constraints according to the business model. For performance reasons, it is recommended to integrate the geospatial engine into the business process and business rule engine. A local GIS engine will provide more functionality and better performance. There is a performance tradeoff between geospatial Web Services calls and GIS local engines.

Geospatial business process, geospatial business rules and geospatial context awareness will provide a formal framework to create complex geospatial processes in many domains such as emergency systems, biology studies, economic development and others. The implementation proposed in this thesis can be applied to any application type and future work may involve design systems in different domains that validate the usability of the geospatial business process system.

REFERENCES

1. Hollingsworth., D., *Workflow Management Coalition: The Workflow Reference Model*. 1995.
2. Aalst, W.M.P.v.d., *The Application of Petri Nets to Workflow Management*. 1998.
3. Nick Russell, A.H.M.t.H., Wil M.P. van der Aalst, *Workflow Control-Flow Patterns: A revised view*. 2006.
4. Nick Russell, A.H.M.t.H., David Edmond, Wil M.P. van der Aalst, *Workflow Data Patterns*. 2004.
5. Nick Russell, A.H.M.t.H., David Edmond, Wil M.P. van der Aalst, *Workflow Resource Patterns*. 2004.
6. Nick Russell, A.H.M.t.H., Wil M.P. van der Aalst, *Workflow Exception Patterns*. 2005.
7. Fu, X., *Analysis of interacting BPEL web services*. 2004.
8. Aalst, W.M.v.d., *Patterns and XPD L: A Critical Evaluation of the XML Process Definition Language*.
9. OMG, *Business Process Model and Notation 2.0 (BPMN)*. 2011.
10. Forgy, C., *Rete: A fast algorithm for the many pattern/many object pattern match problem*. 1982.
11. Dey, A.K., *Understanding and Using Context*. 2006.
12. Matthias Wieland, O.K., Daniela Nicklas, Frank Leyman, *Towards Context-aware Workflows*. 2008.

13. Michael J de Smith, M.F.G., Paul A Longley, *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools*. (Third edition).
14. al, G.C.e., *Towards a generalized map algebra principles and data types*. 2005.
15. Albrecht, J., *Universal analytical GIS operations - a task-oriented systematization of data structure-independent GIS functionality*. 1998.
16. *Google Maps API* <http://code.google.com/apis/maps/index.html>.
17. ESRI, <http://resources.arcgis.com/>.
18. Fielding, R.T., *Architectural Styles and the Design of Network-based Software Architectures*. 2000.
19. Curbera, F., *Unraveling the Web Services Web. An Introduction to SOAP, WSDL, and UDDI*. 2002.
20. Consortium, O.G., *Web Feature Service Implementation Specification*. 2002.
21. Open Geospatial Consortium, I., *OGC® Geography Markup Language 3.3*. 2012.
22. Open Geospatial Consortium, I., *OpenGIS Web Processing Service*. 2007.
23. Michael zur Muehlen, M.I., *Modeling languages for business processes and business rules: A representational analysis*. Information Systems, 2010. **35**(4): p. 379-390.
24. Kovacic, A., *Business renovation: business rules (still) the missing link*, *Business Process Management Journal*. 2004: p. 158.
25. (IBM), C.J.P.a.J.Z., *Non-Functional Requirements in Business Process Modeling*. 2008.

26. Schiefer, J., *Event-Driven Rules for Sensing and Responding to Business Situations*. 2007.
27. Brandon Bennett, A.P.G., *A unifying semantics for time and events*. 2006.
28. Open Geospatial Consortium, I., *OGC® OWS-6 Geoprocessing Workflow Architecture Engineering Report*, 2009.
29. Open Geospatial Consortium, I., *Reference Model for the ORCHESTRA Architecture*. 2007.
30. ESRI, *ArcGIS Model Builder*.
31. Prager, M., *GeoWeb Services Orchestration Based on BPEL or BPMN?* 2009(NESS Slovensko, a.s., Zvolenská cesta 14): p. 10.
32. Albrecht, J., B. Derman, and L. Ramasubramanian, *Geo-ontology Tools: The Missing Link*. Transactions in GIS, 2008. **12**(4): p. 409-424.
33. (OMG), O.M.G., *Object Constraint Language (OCL)*. 2006. **Version 2.0**
34. *Python Shapely Library* <http://pypi.python.org/pypi/Shapely>
35. *Python Projection Library* <http://code.google.com/p/pyproj/>.
36. *Python Web Library* <http://webpy.org/>.
37. jBPM, <http://www.jboss.org/jbpm/>.
38. Drools, <http://www.jboss.org/drools>.
39. Spring, <http://www.springsource.org/>.
40. Solr, <http://lucene.apache.org/solr/>.
41. ExtJS, <http://www.sencha.com/>.

APPENDIX A

GCL USE CASES

Geospatial Business Processes constraint and requirements

- Proximity relationships: Figure 41 shows the first use case using proximity relationships between the tasks of the business process. The process has an associated GCL constraint file which is validated for each process instance. The GeoTask has two parameters which are a unique integer id and the location information in GeoJSON format. The first constraint validates, if the geotask1 and geotask2 are close by at least 95km. There is also validation for the geotask3 and geotask4 to be distant by at least 35km, and the geotask3 and geotask1 to be in the same place. Figure 42 shows the BPMN representation in the map and Figure 43 shows the output of the execution of the process. For this example, the output is the validation check in each task.

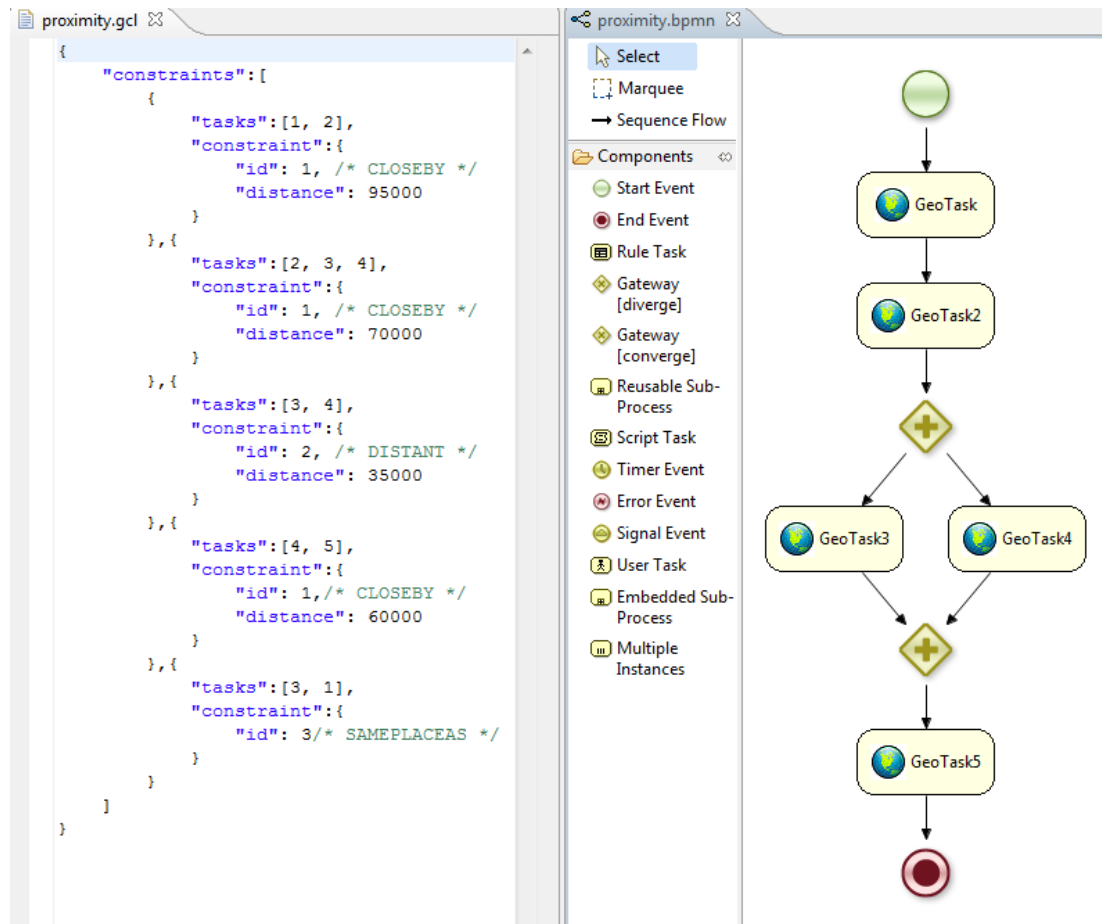


Figure 41: Proximity relationships in BPMN

The output shows the different validations performed by the GCL Engine. As seen in Figure 43, the first task is executed but the validation returns NOCHECK due the rest of the task involved in the constraints are not yet reached. This corresponds to the reachability property for constraints check. When the process reaches geotask2 the first constraint is validated and the second constraint is omitted with NOCHECK. The validation in geotask4 shows different constraints being validated in the same node.

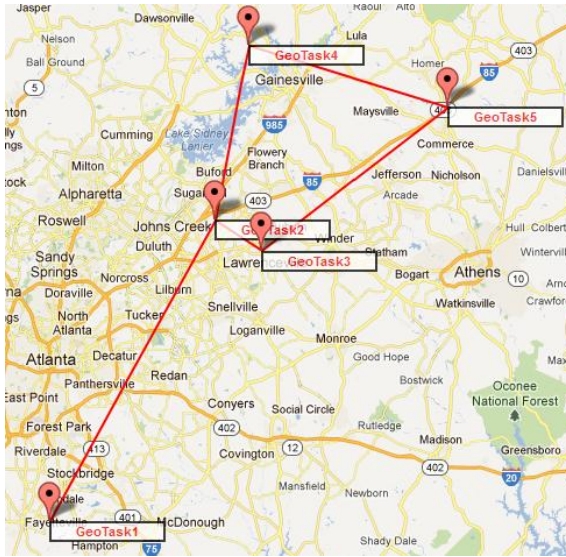


Figure 42: Map representing the proximity constraint in the BPMN process

```

Console
<terminated> ProximityRelationsTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Feb 29, 2012 10:36:35 AM)

-----
Executing Task [id=1, name=GeoTask1]
-----
Validation NOCHECK(reachability property) = GCLConstraint[ name: CLOSEBY, tasks:[1 2 ], params: [95000 ]]
Validation NOCHECK(reachability property) = GCLConstraint[ name: SAMEPLACEAS, tasks:[3 1 ], params: []]
Executing Task:[ GeoTask 1]
-----

-----
Executing Task [id=2, name=GeoTask2]
-----
Validation TRUE = GCLConstraint[ name: CLOSEBY, tasks:[1 2 ], params: [95000 ]]
Validation NOCHECK(reachability property) = GCLConstraint[ name: CLOSEBY, tasks:[2 3 4 ], params: [70000 ]]
Executing Task:[ GeoTask 2]
-----

-----
Executing Task [id=3, name=GeoTask3]
-----
Validation NOCHECK(reachability property) = GCLConstraint[ name: CLOSEBY, tasks:[2 3 4 ], params: [70000 ]]
Validation NOCHECK(reachability property) = GCLConstraint[ name: DISTANT, tasks:[3 4 ], params: [35000 ]]
Validation FALSE = GCLConstraint[ name: SAMEPLACEAS, tasks:[3 1 ], params: []]
Validation Constraints = FALSE
Executing Task:[ GeoTask 3]
-----

-----
Executing Task [id=4, name=GeoTask4]
-----
Validation TRUE = GCLConstraint[ name: CLOSEBY, tasks:[2 3 4 ], params: [70000 ]]
Validation TRUE = GCLConstraint[ name: DISTANT, tasks:[3 4 ], params: [35000 ]]
Validation NOCHECK(reachability property) = GCLConstraint[ name: CLOSEBY, tasks:[4 5 ], params: [60000 ]]
Executing Task:[ GeoTask 4]
-----

-----
Executing Task [id=5, name=GeoTask5]
-----
Validation TRUE = GCLConstraint[ name: CLOSEBY, tasks:[4 5 ], params: [60000 ]]
Validation Constraints = TRUE
Executing Task:[ GeoTask 5]
-----

```

Figure 43: Output of the execution of proximity relationships in BPMN

- Geometric task's relationships: Figure 44 shows the process using geometric task relationships and Figure 46 shows the correspond output. This example uses point, line, and polygon geometries.

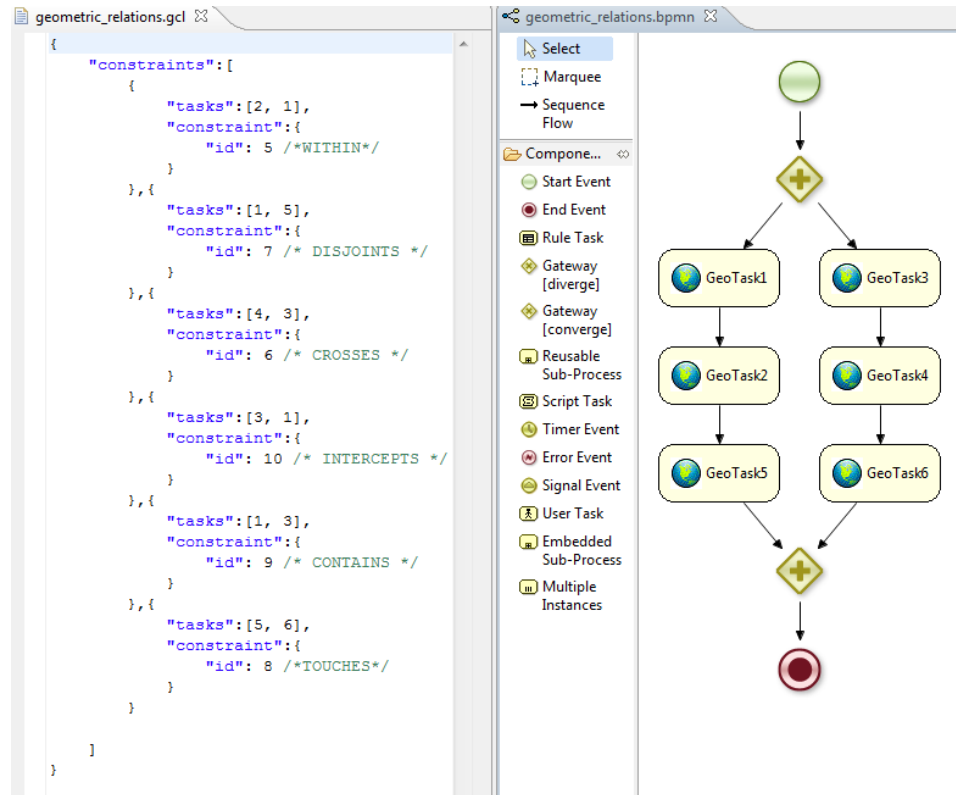


Figure 44: Geometric task's relations in BPMN

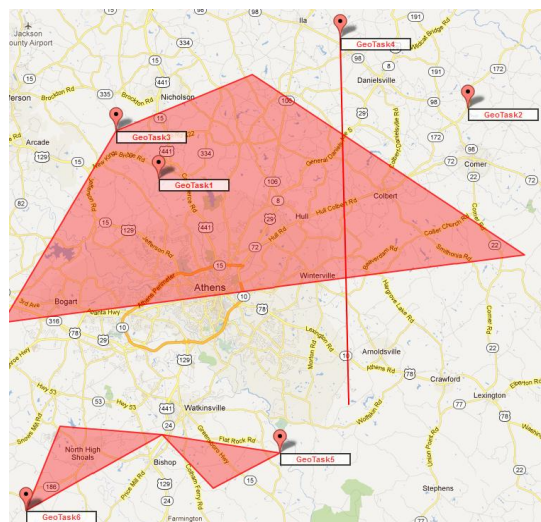


Figure 45: Map representing the geometric constraint in the BPMN process

```
Console X
<terminated> GeometricRelationsTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Feb 29, 2012 10:38:22 AM)

Executing Task [id=1, name=GeoTask1]
-----
Validation NOCHECK(reachability property) = GCLConstraint[ name: WITHIN, tasks:[2 1 ], params: []]
Validation NOCHECK(reachability property) = GCLConstraint[ name: DISJOINTS, tasks:[1 5 ], params: []]
Validation NOCHECK(reachability property) = GCLConstraint[ name: INTERCEPTS, tasks:[3 1 ], params: []]
Validation NOCHECK(reachability property) = GCLConstraint[ name: CONTAINS, tasks:[1 3 ], params: []]
Executing Task:[ GeoTask 1]
-----

Executing Task [id=2, name=GeoTask2]
-----
Validation FALSE = GCLConstraint[ name: WITHIN, tasks:[2 1 ], params: []]
Validation Constraints = FALSE
Executing Task:[ GeoTask 2]
-----

Executing Task [id=5, name=GeoTask5]
-----
Validation TRUE = GCLConstraint[ name: DISJOINTS, tasks:[1 5 ], params: []]
Validation NOCHECK(reachability property) = GCLConstraint[ name: TOUCHES, tasks:[5 6 ], params: []]
Executing Task:[ GeoTask 5]
-----

Executing Task [id=3, name=GeoTask3]
-----
Validation NOCHECK(reachability property) = GCLConstraint[ name: CROSSES, tasks:[4 3 ], params: []]
Validation TRUE = GCLConstraint[ name: INTERCEPTS, tasks:[3 1 ], params: []]
Validation TRUE = GCLConstraint[ name: CONTAINS, tasks:[1 3 ], params: []]
Executing Task:[ GeoTask 3]
-----

Executing Task [id=4, name=GeoTask4]
-----
Validation TRUE = GCLConstraint[ name: CROSSES, tasks:[4 3 ], params: []]
Validation Constraints = TRUE
Executing Task:[ GeoTask 4]
-----

Executing Task [id=6, name=GeoTask6]
-----
Validation TRUE = GCLConstraint[ name: TOUCHES, tasks:[5 6 ], params: []]
Validation Constraints = TRUE
Executing Task:[ GeoTask 6]
```

Figure 46: Output of the execution of geometric relationships in BPMN

For the geometric relationships output one can see that the geotask1 is not within the geotask2. The geometry of geotask3 contains the geometry of geotask1. The geometry of geotask5 touches the geometry of geotask6.

- Task Measurements relationships: Figure 47 shows a BPMN process with measurements constraints. Figure 49 shows the output of the execution of the

process. In this use case the constraints are based on the elevation for the geotask1, area for the geotask2, and length for the geotask3 measurements.

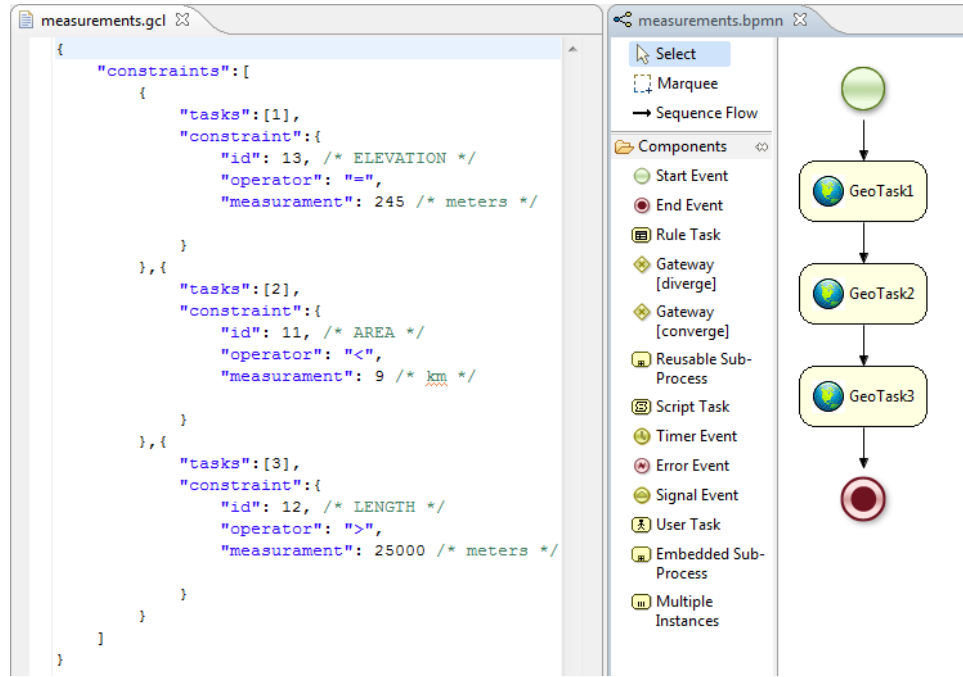


Figure 47: Measurement constraints for BPMN

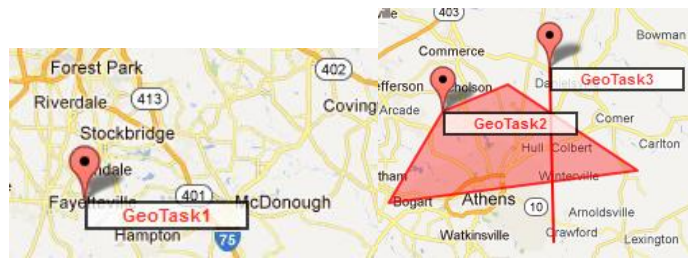


Figure 48: Map representing the measurements constraint in the BPMN process

```
Console X
<terminated> MeasurementRelationsTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Mar 1, 2012 8:55:32 AM)

-----
Executing Task [id=1, name=GeoTask1]
-----
Validation TRUE = GCLConstraint[ name: ELEVATION, tasks:[1 ], params: [= 245 ]]
Validation Constraints = TRUE
Executing Task:[ GeoTask 1]
-----

-----
Executing Task [id=2, name=GeoTask2]
-----
Validation FALSE = GCLConstraint[ name: AREA, tasks:[2 ], params: [< 9 ]]
Validation Constraints = FALSE
Executing Task:[ GeoTask 2]
-----

-----
Executing Task [id=3, name=GeoTask3]
-----
Validation TRUE = GCLConstraint[ name: LENGTH, tasks:[3 ], params: [> 25000 ]]
Validation Constraints = TRUE
Executing Task:[ GeoTask 3]
-----
```

Figure 49: Output of the execution of measurements relationships in BPMN

- Geographic network operations: Figure 50 shows the BPMN process with geographic network constraints. Figure 52 shows the output of the execution of the process. In this set of constraints geotasks are evaluated according to its location to others tasks in a points network. Also the driving time and the driving distance which are part of the road network.

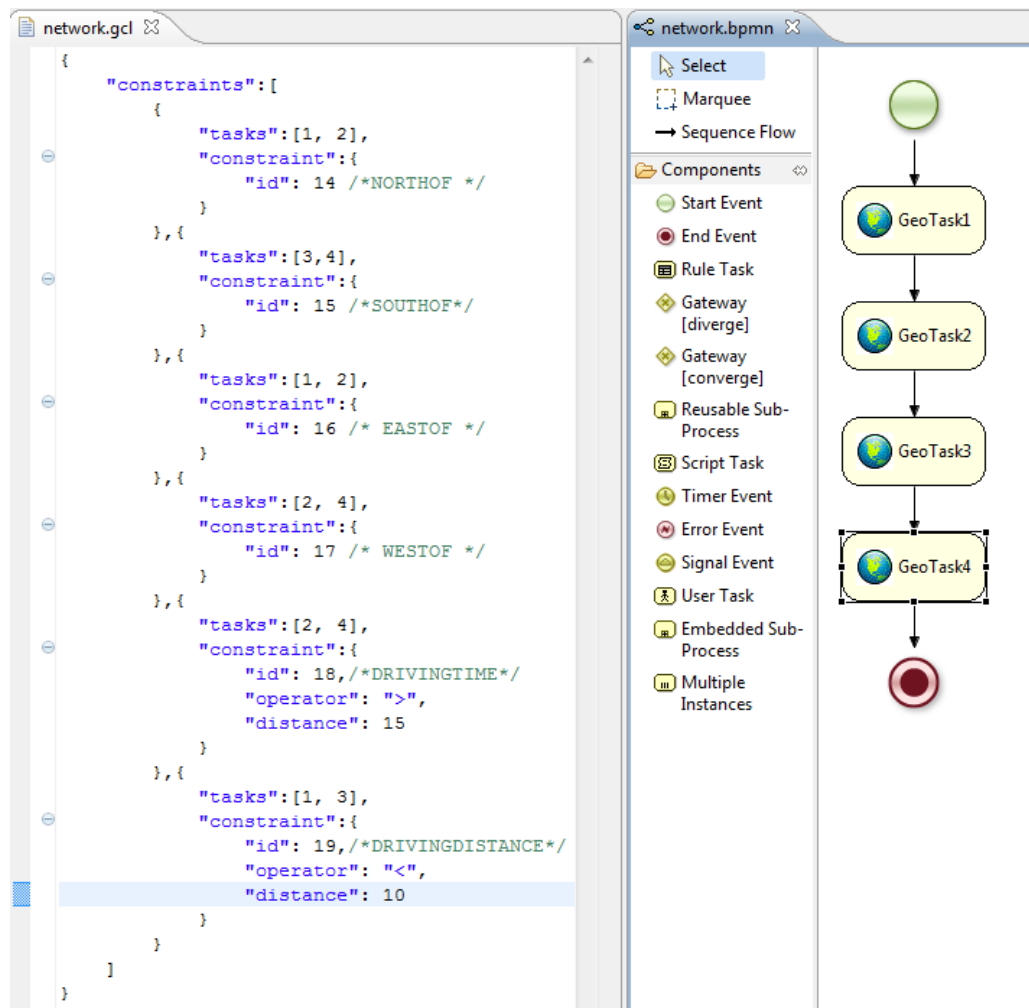


Figure 50: Geographic network relationships in BPMN

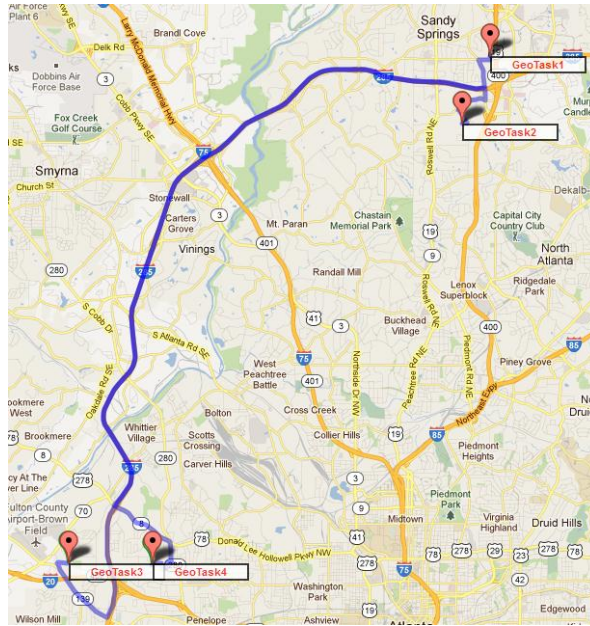


Figure 51: Map representing the network constraint in the BPMN process

```

Console X
<terminated> NetworkRelationsTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Mar 1, 2012 9:03:11 AM)

-----
Executing Task [id=1, name=GeoTask1]
-----
Validation NOCHECK(reachability property) = GCLConstraint[ name: NORTHOF, tasks:[1 2 ], params: []]
Validation NOCHECK(reachability property) = GCLConstraint[ name: EASTOF, tasks:[1 2 ], params: []]
Validation NOCHECK(reachability property) = GCLConstraint[ name: DRIVINGDISTANCE, tasks:[1 3 ], params: [< 10 ]]
Executing Task:[ GeoTask 1]
-----

Executing Task [id=2, name=GeoTask2]
-----
Validation TRUE = GCLConstraint[ name: NORTHOF, tasks:[1 2 ], params: []]
Validation TRUE = GCLConstraint[ name: EASTOF, tasks:[1 2 ], params: []]
Validation NOCHECK(reachability property) = GCLConstraint[ name: WESTOF, tasks:[2 4 ], params: []]
Validation NOCHECK(reachability property) = GCLConstraint[ name: DRIVINGITIME, tasks:[2 4 ], params: [> 15 ]]
Executing Task:[ GeoTask 2]
-----

Executing Task [id=3, name=GeoTask3]
-----
Validation NOCHECK(reachability property) = GCLConstraint[ name: SOUTHOF, tasks:[3 4 ], params: []]
Validation FALSE = GCLConstraint[ name: DRIVINGDISTANCE, tasks:[1 3 ], params: [< 10 ]]
Validation Constraints = FALSE
Executing Task:[ GeoTask 3]
-----

Executing Task [id=4, name=GeoTask4]
-----
Validation FALSE = GCLConstraint[ name: SOUTHOF, tasks:[3 4 ], params: []]
Validation FALSE = GCLConstraint[ name: WESTOF, tasks:[2 4 ], params: []]
Validation TRUE = GCLConstraint[ name: DRIVINGITIME, tasks:[2 4 ], params: [> 15 ]]
Validation Constraints = FALSE
Executing Task:[ GeoTask 4]
-----

```

Figure 52: Output of the execution of geographic network relationships in BPMN

- Geospatial filter relationships: Figure 53 shows a BPMN process with filter constraints based on a constant polygon provided as parameter in the GCL file. Figure 54 shows the output of the process execution. The constraint in this case is applied to the task and the polygon is passed as constant in the GCL file.

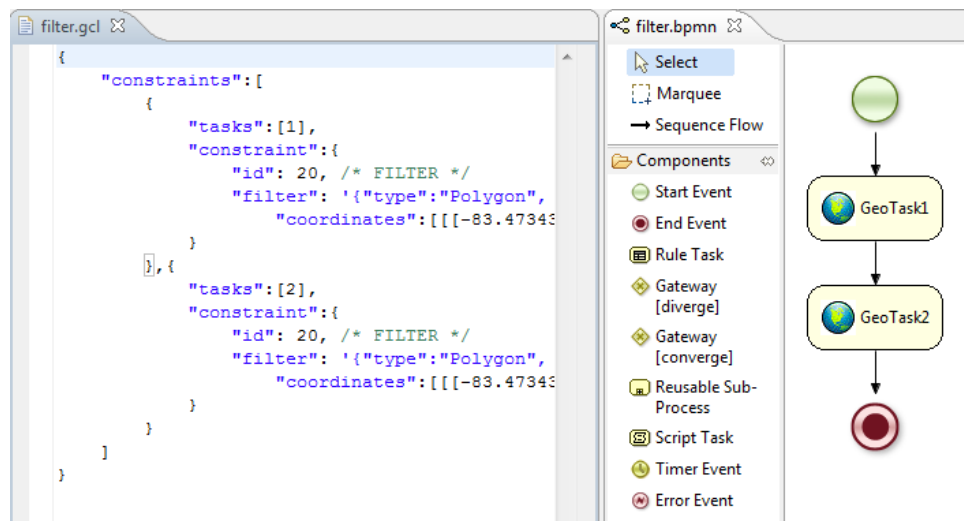


Figure 53: Filter constraint in BPMN

```

Console
<terminated> FilterRelationTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Feb 29, 2012 10:41:14 AM)

-----
Executing Task [id=1, name=GeoTask1]
-----
Validation TRUE = GCLConstraint[ name: FILTER, tasks:[1 ], params: [{"type":"Polygon","coordinates":[[[-83.4734:
Validation Constraints = TRUE
Executing Task:[ GeoTask 1]
-----

Executing Task [id=2, name=GeoTask2]
-----
Validation FALSE = GCLConstraint[ name: FILTER, tasks:[2 ], params: [{"type":"Polygon","coordinates":[[[-83.473:
Validation Constraints = FALSE
Executing Task:[ GeoTask 2]
-----

```

Figure 54: Output of the execution of the filter relationships in BPMN



Figure 55: Map representing the filter constraint in the BPMN process

APPENDIX B

GDL USE CASES

Geospatial Business Rules

- Proximity rules: Figure 56 shows the proximity rules in GDL language.

```
package edu.uga.research.rules

#list any import classes here.
import edu.uga.research.util.model.*;
import edu.uga.research.util.engines.GeoSpatialEngine;

expander proximity.dsl

rule "Close By 200km"
when
    GeoFact1( this closeby[200000] GeoFact2 )
then
    print $g1+" close by 200km or less "+$g2
end

rule "distant by 5km"
when
    GeoFact1( this distant[5000] GeoFact2 )
then
    print $g1+" distant by at least 5km "+$g2
end

rule "sameplaceas"
when
    GeoFact1( this sameplaceas GeoFact2 )
then
    print $g1+" Same place as "+$g2
end

rule "between 5km and 20km "
when
    GeoFact1( this between[5000 20000] GeoFact2 )
then
    print "The distance of "+$g1+" and "+$g2+" is between [5km 10km] "
end
```

Figure 56: Proximity rules in GDL

Figure 57 shows the GeoFacts that are inserted into the business rules knowledge base. After inserting the GeoFacts, the business rule engine fires all the rules and the output can be seen in Figure 58. The output relates all the GeoFacts according to the proximity rules.

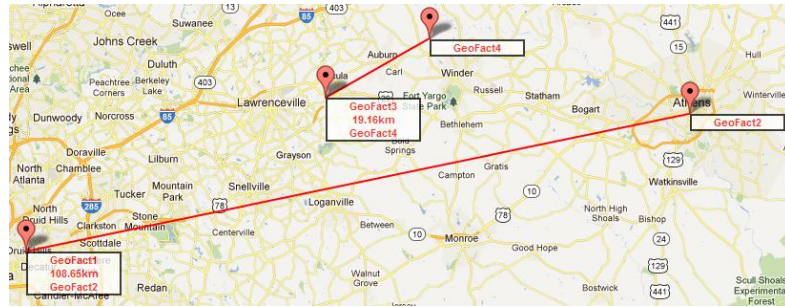


Figure 57: GeoFacts for proximity rules

```

Console X
<terminated> ProximityRulesTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Mar 1, 2012 2:43:05 PM)
Firing proximity rules...
The distance of GeoFact[id=GeoFact4, geometry=Point] and GeoFact[id=GeoFact5, geometry=Point] is between [5km 10km]
GeoFact[id=GeoFact4, geometry=Point] distant by at least 5km GeoFact[id=GeoFact5, geometry=Point]
GeoFact[id=GeoFact4, geometry=Point] close by 200km or less GeoFact[id=GeoFact5, geometry=Point]
GeoFact[id=GeoFact3, geometry=Point] distant by at least 5km GeoFact[id=GeoFact5, geometry=Point]
GeoFact[id=GeoFact3, geometry=Point] close by 200km or less GeoFact[id=GeoFact5, geometry=Point]
GeoFact[id=GeoFact2, geometry=Point] distant by at least 5km GeoFact[id=GeoFact5, geometry=Point]
GeoFact[id=GeoFact2, geometry=Point] close by 200km or less GeoFact[id=GeoFact5, geometry=Point]
GeoFact[id=GeoFact1, geometry=Point] distant by at least 5km GeoFact[id=GeoFact5, geometry=Point]
GeoFact[id=GeoFact1, geometry=Point] close by 200km or less GeoFact[id=GeoFact5, geometry=Point]
The distance of GeoFact[id=GeoFact5, geometry=Point] and GeoFact[id=GeoFact4, geometry=Point] is between [5km 10km]
GeoFact[id=GeoFact5, geometry=Point] distant by at least 5km GeoFact[id=GeoFact4, geometry=Point]
GeoFact[id=GeoFact5, geometry=Point] close by 200km or less GeoFact[id=GeoFact4, geometry=Point]
GeoFact[id=GeoFact5, geometry=Point] distant by at least 5km GeoFact[id=GeoFact3, geometry=Point]
GeoFact[id=GeoFact5, geometry=Point] close by 200km or less GeoFact[id=GeoFact3, geometry=Point]
GeoFact[id=GeoFact5, geometry=Point] distant by at least 5km GeoFact[id=GeoFact2, geometry=Point]
GeoFact[id=GeoFact5, geometry=Point] close by 200km or less GeoFact[id=GeoFact2, geometry=Point]
GeoFact[id=GeoFact5, geometry=Point] distant by at least 5km GeoFact[id=GeoFact1, geometry=Point]
GeoFact[id=GeoFact5, geometry=Point] close by 200km or less GeoFact[id=GeoFact1, geometry=Point]
GeoFact[id=GeoFact3, geometry=Point] distant by at least 5km GeoFact[id=GeoFact4, geometry=Point]
GeoFact[id=GeoFact3, geometry=Point] close by 200km or less GeoFact[id=GeoFact4, geometry=Point]
GeoFact[id=GeoFact2, geometry=Point] distant by at least 5km GeoFact[id=GeoFact4, geometry=Point]
GeoFact[id=GeoFact2, geometry=Point] close by 200km or less GeoFact[id=GeoFact4, geometry=Point]
GeoFact[id=GeoFact1, geometry=Point] distant by at least 5km GeoFact[id=GeoFact4, geometry=Point]
GeoFact[id=GeoFact1, geometry=Point] close by 200km or less GeoFact[id=GeoFact4, geometry=Point]
GeoFact[id=GeoFact4, geometry=Point] distant by at least 5km GeoFact[id=GeoFact3, geometry=Point]
GeoFact[id=GeoFact4, geometry=Point] close by 200km or less GeoFact[id=GeoFact3, geometry=Point]
GeoFact[id=GeoFact4, geometry=Point] distant by at least 5km GeoFact[id=GeoFact2, geometry=Point]
GeoFact[id=GeoFact4, geometry=Point] close by 200km or less GeoFact[id=GeoFact2, geometry=Point]
GeoFact[id=GeoFact4, geometry=Point] distant by at least 5km GeoFact[id=GeoFact1, geometry=Point]
GeoFact[id=GeoFact4, geometry=Point] close by 200km or less GeoFact[id=GeoFact1, geometry=Point]
GeoFact[id=GeoFact2, geometry=Point] distant by at least 5km GeoFact[id=GeoFact3, geometry=Point]
GeoFact[id=GeoFact2, geometry=Point] close by 200km or less GeoFact[id=GeoFact3, geometry=Point]
GeoFact[id=GeoFact1, geometry=Point] Same place as GeoFact[id=GeoFact3, geometry=Point]
GeoFact[id=GeoFact1, geometry=Point] close by 200km or less GeoFact[id=GeoFact3, geometry=Point]
GeoFact[id=GeoFact3, geometry=Point] distant by at least 5km GeoFact[id=GeoFact2, geometry=Point]
GeoFact[id=GeoFact3, geometry=Point] close by 200km or less GeoFact[id=GeoFact2, geometry=Point]
GeoFact[id=GeoFact3, geometry=Point] Same place as GeoFact[id=GeoFact1, geometry=Point]
GeoFact[id=GeoFact3, geometry=Point] close by 200km or less GeoFact[id=GeoFact1, geometry=Point]
GeoFact[id=GeoFact1, geometry=Point] distant by at least 5km GeoFact[id=GeoFact2, geometry=Point]
GeoFact[id=GeoFact1, geometry=Point] close by 200km or less GeoFact[id=GeoFact2, geometry=Point]
GeoFact[id=GeoFact2, geometry=Point] distant by at least 5km GeoFact[id=GeoFact1, geometry=Point]
GeoFact[id=GeoFact2, geometry=Point] close by 200km or less GeoFact[id=GeoFact1, geometry=Point]

```

Figure 58: Output after firing proximity rules

- Geometric relationships rules: Figure 60 shows the geometric relationships rules in GDL language. Figure 59 shows the GeoFacts that are inserted in the knowledge base.

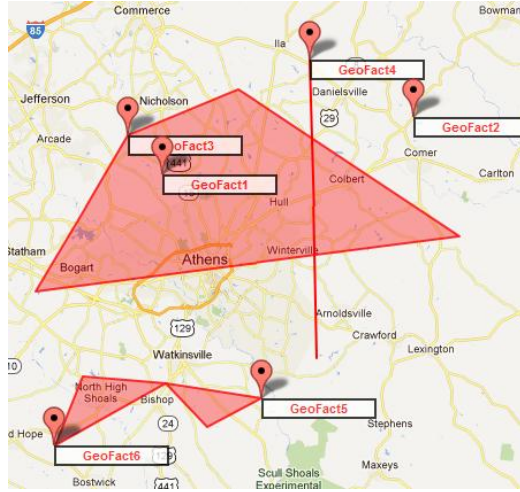


Figure 59: Geo Facts for geometric relationships rules

```
import edu.uga.research.util.model.*;
import edu.uga.research.util.engines.GeoSpatialEngine;

expander geometric_relations.dsl

rule "Within"
  when
    GeoFact1( this within GeoFact2 )
  then
    print $g1+" is within "+$g2
  end

rule "Disjoints"
  when
    GeoFact1( this disjoints GeoFact2 )
  then
    print $g1+" is disjoint "+$g2
  end

rule "Crosses"
  when
    GeoFact1( this crosses GeoFact2 )
  then
    print $g1+" crosses "+$g2
  end

rule "Intercepts"
  when
    GeoFact1( this intercepts GeoFact2 )
  then
    print $g1+" intercepts "+$g2
  end

rule "Touches"
  when
    GeoFact1( this touches GeoFact2 )
  then
    print $g1+" touches "+$g2
  end

rule "Contains"
  when
    GeoFact1( this contains GeoFact2 )
  then
    print $g1+" contains "+$g2
  end
```

Figure 60: Geometric relationships rules in GDL

After inserting the GeoFacts into the knowledge base, the rules engine fires all rules. Figure 61 shows the output of the matching process. As one can see the GeoFact5 and GeoFact6 are matched with the relationships touches. The GeoFact3 contains the GeoFact1. The GeoFact4 crosses the GeoFact3. The GeoFact1 is disjoint of the GeoFact2.

```

<terminated> GeometricRelationsRulesTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Mar 1, 2012 2:46:31 PM)
Firing geoemtric relations rules...
GeoFact[id=Geofact5, geometry=Polygon] touches GeoFact[id=Geofact6, geometry=Polygon]
GeoFact[id=Geofact5, geometry=Polygon] intercepts GeoFact[id=Geofact6, geometry=Polygon]
GeoFact[id=Geofact4, geometry=Line] is disjoint GeoFact[id=Geofact6, geometry=Polygon]
GeoFact[id=Geofact3, geometry=Polygon] is disjoint GeoFact[id=Geofact6, geometry=Polygon]
GeoFact[id=Geofact2, geometry=Point] is disjoint GeoFact[id=Geofact6, geometry=Polygon]
GeoFact[id=Geofact1, geometry=Point] is disjoint GeoFact[id=Geofact6, geometry=Polygon]
GeoFact[id=Geofact6, geometry=Polygon] touches GeoFact[id=Geofact5, geometry=Polygon]
GeoFact[id=Geofact6, geometry=Polygon] intercepts GeoFact[id=Geofact5, geometry=Polygon]
GeoFact[id=Geofact6, geometry=Polygon] is disjoint GeoFact[id=Geofact4, geometry=Line]
GeoFact[id=Geofact6, geometry=Polygon] is disjoint GeoFact[id=Geofact3, geometry=Polygon]
GeoFact[id=Geofact6, geometry=Polygon] is disjoint GeoFact[id=Geofact2, geometry=Point]
GeoFact[id=Geofact6, geometry=Polygon] is disjoint GeoFact[id=Geofact1, geometry=Point]
GeoFact[id=Geofact4, geometry=Line] is disjoint GeoFact[id=Geofact5, geometry=Polygon]
GeoFact[id=Geofact3, geometry=Polygon] is disjoint GeoFact[id=Geofact5, geometry=Polygon]
GeoFact[id=Geofact2, geometry=Point] is disjoint GeoFact[id=Geofact5, geometry=Polygon]
GeoFact[id=Geofact1, geometry=Point] is disjoint GeoFact[id=Geofact5, geometry=Polygon]
GeoFact[id=Geofact5, geometry=Polygon] is disjoint GeoFact[id=Geofact4, geometry=Line]
GeoFact[id=Geofact5, geometry=Polygon] is disjoint GeoFact[id=Geofact3, geometry=Polygon]
GeoFact[id=Geofact5, geometry=Polygon] is disjoint GeoFact[id=Geofact2, geometry=Point]
GeoFact[id=Geofact5, geometry=Polygon] is disjoint GeoFact[id=Geofact1, geometry=Point]
GeoFact[id=Geofact3, geometry=Polygon] intercepts GeoFact[id=Geofact4, geometry=Line]
GeoFact[id=Geofact3, geometry=Polygon] crosses GeoFact[id=Geofact4, geometry=Line]
GeoFact[id=Geofact2, geometry=Point] is disjoint GeoFact[id=Geofact4, geometry=Line]
GeoFact[id=Geofact1, geometry=Point] is disjoint GeoFact[id=Geofact4, geometry=Line]
GeoFact[id=Geofact4, geometry=Line] intercepts GeoFact[id=Geofact3, geometry=Polygon]
GeoFact[id=Geofact4, geometry=Line] crosses GeoFact[id=Geofact3, geometry=Polygon]
GeoFact[id=Geofact4, geometry=Line] is disjoint GeoFact[id=Geofact2, geometry=Point]
GeoFact[id=Geofact4, geometry=Line] is disjoint GeoFact[id=Geofact1, geometry=Point]
GeoFact[id=Geofact2, geometry=Point] is disjoint GeoFact[id=Geofact3, geometry=Polygon]
GeoFact[id=Geofact1, geometry=Point] intercepts GeoFact[id=Geofact3, geometry=Polygon]
GeoFact[id=Geofact1, geometry=Point] is within GeoFact[id=Geofact3, geometry=Polygon]
GeoFact[id=Geofact3, geometry=Polygon] is disjoint GeoFact[id=Geofact2, geometry=Point]
GeoFact[id=Geofact3, geometry=Polygon] contains GeoFact[id=Geofact1, geometry=Point]
GeoFact[id=Geofact3, geometry=Polygon] intercepts GeoFact[id=Geofact1, geometry=Point]
GeoFact[id=Geofact1, geometry=Point] is disjoint GeoFact[id=Geofact2, geometry=Point]
GeoFact[id=Geofact2, geometry=Point] is disjoint GeoFact[id=Geofact1, geometry=Point]

```

Figure 61: Output after firing geometric relationships rules

- Geographic Network relationships rules: This set of rules is divided in two use cases. The first use case as seen in Figure 62 represents the operations over the

GeoFacts point network. Figure 63 shows the GeoFacts inserted in the knowledge base. The second use case represents network operations (driving time and driving distance) over the road network. Figure 66 shows the GDL rules and Figure 64 and Figure 65 shows the GeoFacts inserted into the knowledge base.

```
rule "NorthOf"
when
  GeoFact1( this northOf GeoFact2 )
then
  print $g1+" North of "+$g2
end

rule "SouthOf"
when
  GeoFact1( this southOf GeoFact2 )
then
  print $g1+" South of "+$g2
end

rule "EastOf"
when
  GeoFact1( this eastOf GeoFact2 )
then
  print $g1+" East of "+$g2
end

rule "WestOf"
when
  GeoFact1( this westOf GeoFact2 )
then
  print $g1+" West of "+$g2
end

rule "NorthOf with distance"
when
  GeoFact1( this northOf[7000] GeoFact2 )
then
  print $g1+" North of by less 7km "+$g2
end

rule "SouthOf with distance"
when
  GeoFact1( this northOf[7000] GeoFact2 )
then
  print $g1+" South of by less 7km "+$g2
end

rule "EastOf with distance"
when
  GeoFact1( this eastOf[6000] GeoFact2 )
then
  print $g1+" East by 6km of "+$g2
end
```

Figure 62: Geographic point network relationships rules in GDL

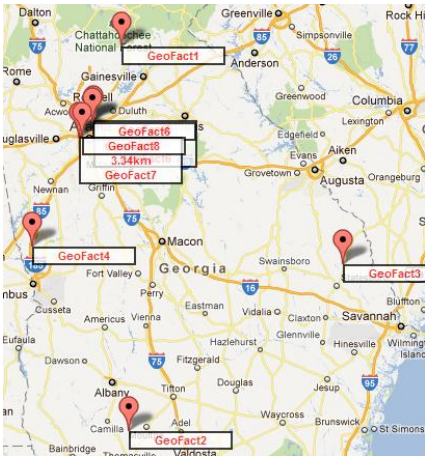


Figure 63: Geo Facts Geographic point network for rules

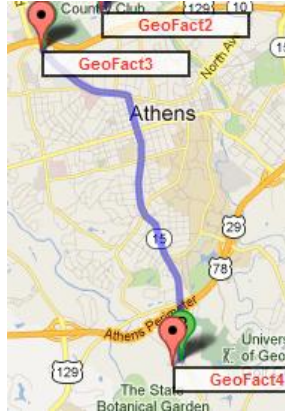


Figure 64: Geo Facts Geographic road network for rules



Figure 65: More Geo Facts Geographic road network for rules

```
package edu.uga.research.rules

#list any import classes here.
import edu.uga.research.util.model.*;
import edu.uga.research.util.engines.GeoSpatialEngine;
expander network.dsl

rule "Driving time range"
when
    GeoFact1( this drivingTime[70 80] GeoFact2 )
then
    print $g1+ " is driving time between 70min and 80min of "+$g2
end

rule "Driving distance range"
when
    GeoFact1( this drivingDistance[4 5] GeoFact2 )
then
    print $g1+ " is driving distance between 4mi and 5mi "+$g2
end
```

Figure 66: Geographic road network relationships rules in GDL

In Figure 67, one can see the output of the first use case. In the output all the GeoFacts are related based on their position to other GeoFacts. Figure 68 shows the output matching for the second use case over the network road.

```

Console
<terminated> NetworkRulesTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Mar 1, 2012 2:50:13 PM)
Firing network rules...
GeoFact[id=Geofact7, geometry=Point] West by 6km of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact7, geometry=Point] West of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact6, geometry=Point] East of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact6, geometry=Point] North of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact5, geometry=Point] East of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact5, geometry=Point] North of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact4, geometry=Point] West of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact4, geometry=Point] South of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact3, geometry=Point] East of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact3, geometry=Point] South of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact2, geometry=Point] East of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact2, geometry=Point] South of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact1, geometry=Point] East of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact1, geometry=Point] North of GeoFact[id=Geofact8, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] East by 6km of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] East of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] West of GeoFact[id=Geofact6, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] South of GeoFact[id=Geofact6, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] West of GeoFact[id=Geofact5, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] South of GeoFact[id=Geofact5, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] East of GeoFact[id=Geofact4, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] North of GeoFact[id=Geofact4, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] West of GeoFact[id=Geofact3, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] North of GeoFact[id=Geofact3, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] West of GeoFact[id=Geofact2, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] North of GeoFact[id=Geofact2, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] West of GeoFact[id=Geofact1, geometry=Point]
GeoFact[id=Geofact8, geometry=Point] South of GeoFact[id=Geofact1, geometry=Point]
GeoFact[id=Geofact6, geometry=Point] East of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact6, geometry=Point] North of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact5, geometry=Point] East of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact5, geometry=Point] North of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact4, geometry=Point] West of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact4, geometry=Point] South of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact3, geometry=Point] East of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact3, geometry=Point] South of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact2, geometry=Point] East of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact2, geometry=Point] South of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact1, geometry=Point] East of GeoFact[id=Geofact7, geometry=Point]
GeoFact[id=Geofact1, geometry=Point] North of GeoFact[id=Geofact7, geometry=Point]

```

Figure 67: Output after firing Geographic network (North of, South of, East of, West of) relationships rules

```

Console
<terminated> NetworkRulesTest2 [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Mar 1, 2012 2:52:54 PM)
Firing network2 rules...
GeoFact[id=Geofact3, geometry=Point] is driving distance between 4mi and 5mi GeoFact[id=Geofact4, geometry=Point]
GeoFact[id=Geofact1, geometry=Point] is driving time between 70min and 80min of GeoFact[id=Geofact4, geometry=Point]
GeoFact[id=Geofact4, geometry=Point] is driving distance between 4mi and 5mi GeoFact[id=Geofact3, geometry=Point]
GeoFact[id=Geofact4, geometry=Point] is driving time between 70min and 80min of GeoFact[id=Geofact1, geometry=Point]
GeoFact[id=Geofact1, geometry=Point] is driving time between 70min and 80min of GeoFact[id=Geofact3, geometry=Point]
GeoFact[id=Geofact3, geometry=Point] is driving time between 70min and 80min of GeoFact[id=Geofact1, geometry=Point]
GeoFact[id=Geofact1, geometry=Point] is driving time between 70min and 80min of GeoFact[id=Geofact2, geometry=Point]
GeoFact[id=Geofact2, geometry=Point] is driving time between 70min and 80min of GeoFact[id=Geofact1, geometry=Point]

```

Figure 68: Output after firing road network relationships rules

- Spatial grouping rules: Figure 69 shows the GDL rules for spatial grouping rules.

There are two basic examples, one using a polygon filter that is inserted in the knowledge base as a constant (not as GeoFact), and the second combining the

spatial filter with a regular expression match over the description of the GeoFact. Figure 70 shows the GeoFacts and the constant polygon inserted into the knowledge base. And Figure 71 shows the output of the execution.

```
package edu.uga.research.rules

#list any import classes here.
import edu.uga.research.util.model.*;
import edu.uga.research.util.engines.GeoSpatialEngine;
expander grouping.dsl

#declare any global variables here
global edu.uga.research.util.model.geometry.Geometry filter

rule "over filter polygon"
  when
    GeoFact( this over area:filter(filter) )
  then
    print $g+" over filter "+filter.toGeoJson()
  end

rule "over search query + polygon"
  when
    GeoFact1( this over area:search( "Hospital", filter ) )
  then
    print $g+" over search Hospital and "+filter.toGeoJson()
  end
```

Figure 69: Spatial grouping relationships rules in GDL



Figure 70: Geo Facts and polygon constant spatial grouping for rules

```

Console X
terminated> GroupingRulesTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Mar 1, 2012 2:54:39 PM)
Firing grouping rules...
GeoFact[id=Geofact5, geometry=Point] over search Hospital and {"type":"Polygon","coordinates":[[[-83.83598,34
GeoFact[id=Geofact5, geometry=Point] over filter {"type":"Polygon","coordinates":[[[-83.83598,34.14136],[-83.
GeoFact[id=Geofact4, geometry=Point] over filter {"type":"Polygon","coordinates":[[[-83.83598,34.14136],[-83.
GeoFact[id=Geofact1, geometry=Point] over search Hospital and {"type":"Polygon","coordinates":[[[-83.83598,34
GeoFact[id=Geofact1, geometry=Point] over filter {"type":"Polygon","coordinates":[[[-83.83598,34.14136],[-83.

```

Figure 71: Output after firing spatial grouping relationships rules