

COMMUNICATIONS SERVER APPARATUS AND METHOD FOR PROCESSING SERVICE
REQUESTS

Technical Field

- 5 The invention relates generally to the field of communications. One aspect of the invention relates to a communications server apparatus for processing service requests. Another aspect of the invention relates to a method, performed in a communications server for processing service requests. Another aspect of the invention relates to a computer program product comprising instructions therefor. Another aspect of the
- 10 invention relates to a computer program comprising instructions therefor. Another aspect of the invention relates to a non-transitory storage medium storing instructions therefor. Another aspect of the invention relates to a communications system for implementing a service, including a communications server for processing service provision requests.
- 15 One aspect of the invention has particular, but not necessarily exclusive, application in a transport (e.g., taxi and/or delivery) service.

Background

- Pricing strategies for services such as transport (taxi) services can be complex. Whilst
- 20 there may, of course, be a basic fare based on a standard tariff plus a mileage rate based on a fee per unit of distance (km or miles), there may be a number of other factors which will affect the price of a journey under certain conditions. Thus, the standard tariff and/or the price per unit distance may change depending on the time of day or, indeed, the total distance required to be travelled. One or both of these factors may also change
- 25 depending on the time it takes to complete the journey. Different discounts may be required to be provided under certain conditions and, indeed, various surcharges may also be required to be applied depending on certain conditions. In prior art systems, a programmer is required to write a separate conditional subroutine (i.e. 'mini programs') for each and every permutation of the pricing strategy. All of these subroutines are
- 30 compiled into machine executable code and the associated machine-executable code is stored in the memory. and executed when called by the main routine in response to an associated condition being 'true'.

There are a number of issues associated with this method of developing, maintaining and reviewing pricing strategies for services. One particular issue is the need to take the system off-line in order to compile a changed or added conditional price calculation to the overall pricing strategy. A programmer is required to enter the respective conditional subroutine in the programming language being used to implement the system, and then compile the subroutine, as described above. The new machine-executable code is stored in the memory, before the system can go back on-line. Clearly, for on-demand services that generally operate 24 hours a day, 7 days a week, it is undesirable for the system to be off-line for the period of time required to implement such a change. Furthermore, the need to employ a programmer to implement the change is inconvenient and can incur unnecessary further delay and costs. The programming language used to generate pricing strategies may not be sufficiently intuitive, such that it takes some skill and effort to define a subroutine that performs a relatively complex conditional pricing calculation, such that errors can easily be made and the methods used are not readily transferrable to pricing strategies for other services, which would require a completely new system to implement.

Another issue arises in the manner in which the resultant executable code is stored and subsequently utilised. Each time a particular condition is met, the system calls the appropriate machine-executable code to perform the respective subroutine and output a price. However, as the number of conditional pricing calculations increases, so too does the processing and memory overhead required to implement a pricing strategy.

In some prior art arrangements, a master fare formula and numerous fare modifiers are used to implement a complete pricing strategy which, in itself, is a limiting factor. Modifying and extending the pricing strategy is complex and time-consuming, and any modifications have to be rigorously reviewed and tested for regression, which reduces engineering velocity. Furthermore, in a transport system, for example, each vehicle level configuration affects how the corresponding fare modifiers alter a fare and different configurations lead to diverse and complex price calculations which make it challenging for any non-programmers to understand the pricing strategy employed.

US2006/0155654 describes an editor for defining and editing pricing calculations, and also a system for automatic costing of a service. The editor, provided as part of a user interface, enables a user to enter a pricing calculation in the form of a tree structure using a library of pre-stored nodes connected by arcs. The resultant pictorial representation of the pricing strategy and price calculations still requires a significant amount of skill to create with any accuracy. Each price calculation is, effectively, a separate entity (or a 'branch' of the tree) such that a complex pricing strategy, which comprises many different conditional pricing calculations, will accordingly result in a very large tree structure. The tree must be created or edited whilst the system is off-line (which requires undesirable 'down time') so that it can be transformed into JAVA code for processing by a JAVA virtual machine, and any changes to the tree would require complete regeneration of the JAVA code.

Summary

Aspects of the invention are as set out in the independent claims. Some optional features are set out in the dependent claims.

Implementation of the techniques disclosed herein may have significant technical advantages. Implementations of the claimed apparatus incorporate an AST interpreter directly into its run-time. This allows users to add and/or modify arithmetic expressions within the pricing strategy, and save them to the formula database, whilst the system remains live and running. When such an arithmetic expression is to be applied to perform a price calculation (e.g. because the type of service requested is associated with the given arithmetic expression for determining the service price), it is then parsed and converted into a representative AST, again whilst the service remains live and running (and simultaneously calculating prices for other service provision requests). Furthermore, once an arithmetic expression has been parsed and converted into an AST representation once, the respective AST is cached and it can then be used again when the same price calculation is to be performed again. This aspect offers a clear and significant technical advantage in that, not only is the need for down time of the system eliminated, but the processing overhead is massively reduced when compared with prior art systems.

The string-based arithmetic expressions are intuitive and understandable, and do not require a user to be skilled in computer science and programming concepts such as tree construction. This can potentially significantly increase the number of price calculations that can be incorporated into a pricing strategy. Arithmetic expressions of formulae can be constructed by using or combining previously-defined formulae with new expressions, which provides clear technical advantages in that the need to construct a tree structure for each price calculation expression (as required in prior art systems) is eliminated. The re-usability of the arithmetic expressions results in significantly lower memory and storage usage. It also provides greater consistency across sets of formulae (because common computations can be defined as sub-formulae and used to compose other formulae). Furthermore, it results in a lower immediate complexity because complex formulae can be abstracted into sub-formulae.

The arithmetic expressions can not only be re-used, but once they have been called and used once, the representative AST is cached, and this can also be re-used every time the respective arithmetic expression is to be applied and until it requires updating, which minimises processing time and enables efficient processing of large numbers of separate price calculations concurrently and in real time.

Brief Description of the Drawings

The invention will now be described by way of example only, and with reference to the accompanying drawings in which:

Figure 1 is a schematic block diagram illustrating an exemplary communications system including a communications server apparatus for generating and implementing pricing strategies for service provision requests related to a service;

Figure 2 is a schematic block diagram illustrating modules of a price calculation engine of a communications server apparatus for generating and implementing pricing strategies for service provision requests related to a service;

Figure 3 is a schematic diagram illustrating a user interface display of the editing function of a communications server apparatus for generating and implementing pricing strategies for service provision requests related to on-demand service provision;

Figure 4 is a schematic process diagram illustrating an exemplary method for implementing a price calculation for a specified service provision request;

Figure 4A illustrates a parse tree representative of an example arithmetic price calculation expression;

- 5 Figure 4B illustrates a Symbol Table for use in a price calculation process for the arithmetic expression illustrated in Figure 4A; and

Figure 5 is a schematic representation of a process for generating and implementing a pricing strategy for service provision requests related to on-demand service provision.

10

Detailed Description

- The techniques described herein are described primarily with reference to transport services, such as taxi, parcel/document delivery or food delivery services, but it will be appreciated that these techniques may have a broader reach and cover other types of
- 15 services wherein multiple variable quantum modifiers may affect price (or quantum) calculations and there is a need to generate and implement a pricing strategy comprising multiple different price calculations in respect of a specified service provision.

- The described techniques may be applied to any context in which pricing calculations are
- 20 to be performed to obtain prices for products and services. As non-limiting examples, the described techniques may be used for transport services (e.g. ride-hailing services, ride-sharing services, etc.), logistics-related services (e.g. delivery of documents, packages, parcels, goods, etc.), finance-related services (e.g. payment, banking, insurance, etc.), food-related services (e.g. food ordering, food delivery, etc.), goods-related services (e.g.
- 25 purchase and/or delivery of groceries, packaged goods/foods, household products, etc.), and trade or handyman-related services (e.g. plumbing services, pest control services, etc.)

- Referring first to Figure 1, a communications system 100 is illustrated. Communications
- 30 system 100 comprises communications server apparatus 102, user communications device 104 and service provider communications device 106. These devices are connected in the communications network 108 (for example the Internet) through respective

communications links 110, 112, 114 implementing, for example, internet communications protocols. Communications devices 104, 106 may be able to communicate through other communications networks, such as public switched telephone networks (PSTN networks), including mobile cellular communications networks, but these are omitted from Figure 1
5 for the sake of clarity.

Communications server apparatus 102 may be a single server as illustrated schematically in Figure 1, or have the functionality performed by the server apparatus 102 distributed across multiple server components. In the example of Figure 1, communications server
10 apparatus 102 may comprise a number of individual components including, but not limited to, one or more microprocessors 116, a memory 118 (e.g., a volatile memory such as a RAM) for the loading of executable instructions 120, the executable instructions defining the functionality the server apparatus 102 carries out under control of the processor 116. Communications server apparatus 102 also comprises an input/output
15 module 122 allowing the server to communicate over the communications network 108. User interface 124 is provided for user control, including configuration and manipulation of price calculation formulae within the pricing strategy for the service provision, and may comprise, for example, computing peripheral devices such as display monitors, computer keyboards and the like. Communications server apparatus 102 also comprises a database
20 126, the purpose of which will become readily apparent from the following discussion. In this embodiment, database 126 is part of the communications server apparatus 102, however, it should be appreciated that database 126 can be separated from communications server apparatus 102 and database 126 may be connected to the communications server apparatus 102 via communications network 108 or via another
25 communications link (not shown). The communications server apparatus 102 also comprises a Price Calculation Engine (PCE) 202, which will be described in more detail hereinafter.

User communications device 104 may comprise a number of individual components
30 including, but not limited to, one or more microprocessors 128, a memory 130 (e.g., a volatile memory such as a RAM) for the loading of executable instructions 132, the executable instructions defining the functionality the user communications device 104

carries out under control of the processor 128. User communications device 104 also comprises an input/output module 134 allowing the user communications device 104 to communicate over the communications network 108. User interface 136 is provided for user control. If the user communications device 104 is, say, a smart phone or tablet
5 device, the user interface 136 will have a touch panel display as is prevalent in many smart phone and other handheld devices. Alternatively, if the user communications device is, say, a desktop or laptop computer, the user interface may have, for example, computing peripheral devices such as display monitors, computer keyboards and the like.

10 Service provider communications device 106 may be, for example, a smart phone or tablet device with the same or a similar hardware architecture to that of user communications device 104. Service provider communications device 106 may comprise a number of individual components including, but not limited to, one or more microprocessors 138, a memory 140 (e.g., a volatile memory such as a RAM) for the loading of executable
15 instructions 142, the executable instructions defining the functionality the service provider communications device 106 carries out under control of the processor 138. Service provider communications device 106 also comprises an input/output module (which may be or include a transmitter module/receiver module) 144 allowing the service provider communications device 106 to communicate over the communications network 108. User
20 interface 146 is provided for user control. If the service provider communications device 106 is, say, a smart phone or tablet device, the user interface 146 will have a touch panel display as is prevalent in many smart phone and other handheld devices. Alternatively, if the service provider communications device is, say, a desktop or laptop computer, the user interface may have, for example, computing peripheral devices such as display
25 monitors, computer keyboards and the like.

In one embodiment, the service provider communications device 106 is configured to push data representative of the service provider (e.g., service provider identity, location and so on) regularly to the communications server apparatus 102 over communications
30 network 108. In another, the communications server apparatus 102 polls the service provider communications device 106 for information. In either case, the data from the service provider communications device 106 are communicated to the communications

server apparatus 102 and at least some parameters or characteristics thereof stored in relevant locations in the database 126.

5 In one embodiment, the user communications device 104 is configured to push data representative of the user (e.g., location, required pick-up times, destination, selected vehicle type, customer details, and so on) regularly to the communications server apparatus 102 over communications network 108. In another, the communications server apparatus 102 polls the user communications device 104 for information. In either case, the data from the user communications device 104 (also referred to herein as service
10 requests or service provision requests) are communicated to the communications server apparatus 102 and at least some parameters or characteristics thereof stored in relevant locations in the database 126.

The communications server apparatus 102 is configured to implement a pricing strategy in
15 relation to service provision requests. In an implementation, the service may be a transport service, and the pricing strategy comprises multiple different fare calculations, selected according to certain conditions and/or characteristics or parameters of the service request (i.e., a request for transportation). As discussed above, prior art systems need a programmer to add or modify price calculation subroutines, and the coded
20 subroutines then require an off-line compilation process in order for the new or modified calculation to become “live” in the pricing strategy. As such, each time a fare calculation needs to be added or changed, there is some ‘down time’ as the system is taken off-line while a skilled programmer adds or changes the relevant subroutine and performs a full compilation of the new (complete) program defining the pricing strategy. Not only is this
25 ‘down time’ highly undesirable for a service provision that operates 24 hours a day, 7 day a week, but it also requires significant processing and memory overhead to re-compile a large (and often unwieldy) piece of software code to incorporate the new or modified fare calculation, which is exacerbated by the inevitable code duplication that occurs as a result of this process. The subroutines may be conditional. In other words, if one or more
30 certain conditions are met, then a respective subroutine for implementing the associated fare calculation is called and executed using values derived or calculated from the service

provision request data (e.g., distance, estimated time) and (optionally) any other relevant parameters, which also have to be called or fetched from a memory layer. This, too, increases the memory and processing overhead required to implement the system which, ultimately requires additional memory and processing overhead to be added, at significant expense, to avoid causing the system to be so slow that it cannot operate effectively in real time, or it requires the price strategies to be limited to a finite number of fare calculations, which stifles growth and limits the extent and services that can be administered by the system.

Implementations of the techniques disclosed herein seek to address at least some of these issues by utilising an entirely different method of adding and modifying price calculations in a pricing strategy that does not require any 'down time', and also utilising a unique caching process for ensuring that unnecessary processing is avoided and the system can continue to function in real time, almost irrespective of the size and complexity of the pricing strategy. An added benefit is that non-programmers can, intuitively, enter and modify price calculations and also clearly visualise the overall pricing strategy as it fits in with the wider business context.

Referring to Figure 2 of the drawings, the communications server apparatus is configured to operate a price calculation engine 202 as a microservice incorporating the following modules: an editing function 204 supporting a pricing domain language for configuring and manipulating price calculations in the form of arithmetic expressions, a validation module 206 configured to validate the arithmetic expressions entered by a user, a formula database 208 which stores all of the arithmetic expressions making up a complete pricing strategy for a service provision, a calculation framework 210 configured to perform all actions required to calculate a price for a service provision request, including data retrieval, communication with external services, various validations and other computations (as will be described in more detail later), and a parser 212 configured to translate the above-referenced arithmetic expressions into respective in-memory representations that can be understood and processed by the calculation framework.

The communications server 102 supports a declarative programming ‘language’ (pricing domain language) for use in defining price calculations and implementing a pricing strategy in relation to service provision requests, for example in a shared economy on-demand taxi service.

5

Referring to Figure 3 of the drawings, in an editing mode, the user of the communications server 102 is presented (by the editing function 204, Figure 2) with an editing page 302 via the user interface (124 – Figure 1). Using any suitable input means (e.g., a keyboard), a user can enter a new string/grammar-based arithmetic expression representative of a particular price calculation. The editing page 302 and the arithmetic expressions (and the functionality by which they can be created and edited) act, essentially, as a ‘façade’ over the price calculation framework functionality (202 – Figure 2) operating ‘behind’ it. The editing mode can be utilised whilst the service provision system is on-line, and all other existing price calculations can remain active and useable whilst the user adds or edits another price calculation expression.

10

15

In this example, the editing page 302 includes a formula display area 304 showing the formula being edited in textual form, as a string-based expression. The page further includes a selection interface 306 for selecting existing (previously defined) named formulae to be incorporated into the edited formula (e.g. in this example a previously defined formula “StandardFare” has been added as an operand of the new formula). The page further provides a selection interface 308 for selecting pricing components/symbols to be added to the formula (e.g. predefined coefficients/constants), a selection interface 310 for selecting formula blocks such as functions to be added to the formula (e.g. a MAX function which selects a maximum value from two or more values), and a selection interface 312 for selecting operators (e.g. addition, multiplication etc.) to be added to the formula. The selection interfaces may be implemented as icon sets, lists, drop down boxes or in any other appropriate way. The user can drag elements from the selection interfaces 306, 308, 310, 312 to formula display region 304 to add them to the edited formula.

20

25

30

Thus, the string-based expressions for the formulae are displayed in real time on the editing page 302 as they are created and edited, using the pricing domain language (PDL). Alternatively or additionally, the interface may also support direct text input of a PDL formula into formula display area 304. A name of the formula is displayed (and may be edited) in name field 314. Examples of price calculation expressions using PDL are given below.

Examples of PDL Formulae for Public Transportation Services

Public transportation services such as taxis usually operate with metered or flat pricing strategies. These pricing strategies are generally “static” in nature. This means that the fares do not change due to dynamic factors such as traffic conditions, passenger demand, or driver supply. For example, a passenger is likely to pay the same fare using the transportation service to travel from their home to office every Tuesday at 8 am.

Metered: Fares are charged according to the taximeter and applicable surcharges. Some examples of metered pricing strategies for public transportation services are provided in the following PDL formulae.

$$\text{MeteredFare} = \text{Distance} * \text{MeteredCharges}$$

$$\text{TaxiFare} = \text{FlagFare} + \text{MeteredFare} + \text{BookingFees} + \text{PeakHourSurcharge} + \text{RoadTaxes}$$

Flat: Fares are charged according to a flat rate, regardless of distance or traffic conditions. We can describe an example of this pricing strategy for public transportation services with the following PDL formula.

$$\text{FlatFare} = 80$$

Examples of PDL Formulae for On-Demand Transportation Services

On-Demand Transportation services operate with similar pricing strategies as conventional public transportation services described in the previous section. However, their pricing strategies are more “dynamic” in nature. For instance, on-demand services may incorporate dynamic components such as “Demand Charges”, “Dynamic Surcharges”, “Dynamic Discounts”, and “Location Fees”, which take traffic conditions, passenger demand, driver supply, and other real-time factors into account when pricing a ride. The following PDL formulae provide some examples of pricing strategies for on-demand transportation services.

10
$$\text{StandardFare} = \text{BasicFare} + \text{Distance} * \text{DistanceCharges} + \text{Duration} * \text{DurationCharges}$$

$$\text{DemandAdjustedFare} = (\text{StandardFare} * \text{Max}(\text{DemandCharges}, \text{MinDemandCharges}) + \text{DynamicSurcharges} - \text{DynamicDiscounts}) * \text{LocationFees}$$

15
$$\text{TieredDemandAdjustedFare} = (\text{TieredFare} * \text{Max}(\text{DemandCharges}, \text{MinDemandCharges}) + \text{DynamicSurcharges} - \text{DynamicDiscounts}) * \text{LocationFees}$$

Whilst multiple expressions are shown concurrently above, it is envisaged that, in normal use, a user may only work on, and display, one expression at a time in the Figure 3 interface. The expressions shown above are provided purely by way of example, and are in no way intended to limit the scope of the invention. Many different terminologies may be employed, depending on the business context in which the system is employed, and many different arithmetic expressions are envisaged with many different operands and operators, and the invention is in no way intended to be limited in this regard.

It will be understood that when a customer submits a service provision request (via the customer communications device 104), the request will include data representative of the pickup and dropoff locations, and a selected vehicle type, at least. Thus, a value for a parameter ‘Distance’ can be calculated by the communications server apparatus 102 using location data and/or satellite navigation data, wherein ‘Distance’ is the distance between the pickup and dropoff locations (in any suitable unit, e.g. km). When a service provision

request is received, the value for 'Distance' is recorded in a state (symbol) table. The time it is expected to take to fulfil the service provision request can also be determined using satellite navigation data and recorded as a value denoted 'Duration' in the state (or symbol) table (again in any suitable unit, e.g. minutes). A predefined (but editable) per unit coefficient, 'DistanceCharges', may be set, for example, as a \$/km price calculator, and another predefined (but, again, editable) coefficient, 'DurationCharges', may also be set, for example, as a \$/minute price calculator. These coefficients are pre-set, and periodically changed, to match business needs to ensure that the pricing strategy takes account of all aspects of a service provision request. As shown in the second example above, a pre-set coefficient, 'BasicFare' may also be stored as a monetary value representative of a standard tariff (chargeable irrespective of distance and/or time taken).

As shown above, and purely by way of example, various types of price calculations may be defined to support different fare calculations, such as MeteredFare, TaxiFare and FlatFare in the first example and StandardFare, DemandAdjustedFare, and TieredDemandAdjustedFare in the second example. All of these price calculation types are intuitively named and defined, and can be made specific to the particular service provision, as required. Furthermore, as the examples illustrate, one price calculation formula may be defined in terms of another (e.g. DemandAdjustedFare references StandardFare).

Furthermore, a formula may reference fixed values. For example, StandardFare is an arithmetic expression defined in terms of the operands, BasicFare, Distance, DistanceCharges, Duration and DurationCharges, which are, once again, intuitively-named descriptors of the values or variables they denote. When executed, Distance and Duration will be obtained from the service provision request. BasicFare, DistanceCharges and DurationCharges are, as described above, pre-set values stored in the memory (126 – Figure 1), and may be periodically changed, as required, by an administrator to reflect commercial (and/or other) changes and needs.

DemandAdjustedFare is defined in terms of the above-referenced StandardFare and another set of variables comprising DemandCharges, MinDemandCharges,

DynamicSurcharges, DynamicDiscounts and LocationFees. Thus, the administrator does not have to re-enter the expression for StandardFare to define DemandAdjustedFare. Instead, the entire expression can be included in the new expression as an operand. Once again, the new constants and variables defined in the expression for DemandAdjustedFare

5 denote (editable) values that are stored in association therewith in the memory and/or are obtained or computed dynamically as needed. For example, DynamicSurcharges, DynamicDiscounts, and LocationFees are values that are calculated dynamically and stored in memory once per request when the price calculation engine receives a service request, and they are computed taking traffic conditions, passenger demand, driver supply, and other real-

10 time factors into account. Other such variables may be updated less frequently (e.g. hourly or daily) by external processes, and/or may be obtained as needed by calls to an appropriate API (application programming interface), as described in more detail later. Parameters of a price calculation may thus include values derived directly from the service request (e.g. Distance), and values (whether constant/user-configurable or dynamically computed) derived from some

15 external source, independently of the service request (e.g. currently applicable surcharges).

TieredDemandAdjustedFare similarly, re-uses a previously-defined expression 'TieredFare' as an operand, and references other named constants or variables that can be retrieved from the memory or determined from the service provision request if the price calculation

20 TieredDemandAdjustedFare is invoked. FlatFare (in the first example), in contrast, is simply defined as a numerical constant (which can be changed as required).

The expressions can be written and modified quickly, and are much less prone to the misconfigurations and bugs common in known systems. The relatively limited language

25 that can be used to configure and manipulate the above-referenced price calculation framework sitting 'behind' the user interface allows various stakeholders to understand intuitively the code and framework underpinning the essential parts of the business. Multiple such formulae can be defined to cater for pricing strategies for different service provisions (e.g., transport, food and document/parcel delivery).

30

The editor function (204 – Figure 2) allows users to create pricing calculation formulae by using customised, business-specific arithmetic expressions. As will be described

hereinafter, a parser can convert such arithmetic expressions into abstract syntax trees (ASTs) using the defined grammar. This grammar-based approach allows users to essentially create new programming 'languages' specific to their business. New price calculation formulae can be built using pre-defined price calculation formulae (i.e., user-constructed formulae can depend on other user-constructed formulae). This eliminates unnecessary complexity and code duplication, in contrast to prior art systems which require a user to create a new tree structure or subroutine for each and every price calculation of a pricing strategy. Once a price calculation formula of a type such as that illustrated in Figure 3 is created, a price calculation engine performs a validation process to determine if the arithmetic expression meets a small number of inherent rules, namely that the operands are defined somewhere in the system and that the arithmetic syntax of the expression is correct. This validation process is made very simple and quick because arithmetic expressions are so intuitive and straightforward to build, that it is much less likely that an error will be made when compared with programming a subroutine or building a tree-like structure of nodes. If an operand is used that has not been previously defined or an error in the arithmetic syntax is identified, the price calculation engine returns an error message, allowing the user to resolve the issue. Once validated, a price calculation expression can be stored as a named formula in a formula database (208 – Figure 2) for future use. The system can remain on-line throughout this process, and each validated formula stored in the formula database is available for immediate use by the system without it first needing to be compiled.

The formula database 208 is a storage layer that stores the string/grammar-based arithmetic calculation formulae. Formulae for different vehicle types, for example, can be stored in a relational database as strings. The AST representations for these formulae become heavily cached (due to volume of usage), so a string-based arithmetic expression does not, by any means, need to be parsed whenever a fare (or other price, depending on the service provision) needs to be calculated. However, whenever there is a change in a formula, the corresponding cached data would be invalidated or updated. Similarly, a new formula would need to be parsed once when it is called, but after that the cached AST representation could be used for every subsequent call, until the formula is changed and

the cached data is invalidated and needs to be updated. This process is described in more detail below.

Referring back to Figure 1 of the drawings, the communications server apparatus 102 is configured, under control of the processor 116, to execute instructions 120 stored in the memory 118 to perform a method as illustrated schematically in the flow diagram of Figure 4. Thus, and referring additionally to Figure 4 of the drawings, upon receipt of a service provision request 402 (e.g., a request for a taxi from location A to location B at a specified time, and specifying a vehicle type), the communications server 102 submits (at step 404) a request to the price calculation engine (202 – Figure 2) to calculate a price for the service provision.

The service request for a price calculation contains a unique identifier of a particular price calculation formula stored in the formula database. In an embodiment, the identifier includes the formula name and a formula version identifier (e.g. version number), to allow multiple different versions of a formula to coexist in the system. The system retrieves the corresponding pricing formula from the database (or its cached AST representation, as described below) and calculates the price according to the formula.

The price calculation formula selected to calculate the price will, of course, be dependent on the specific characteristics of the service provision request, but in this case, consider the price calculation formula given in the example above:

$$\text{StandardFare} = \text{BasicFare} + \text{Distance} * \text{DistanceCharges} + \text{Duration} * \text{DurationCharges}$$

The selected price calculation formula (i.e., a user-defined arithmetic expression of the type described above with reference to Figure 3) is fetched from the formula database (208 – Figure 2) and the price calculation framework (212 – Figure 2) performs two processes concurrently. In a first such process, it determines (at step 406) the set of pricing coefficients utilised in the selected price calculation formula and then checks a caching layer to determine if values for that set of coefficients have previously been cached. If not, it fetches the coefficient values from a database (126 – Figure 1) and updates the cache (at step 408) with those values (so that they can be more quickly accessed the next time the selected price calculation formula is selected for use). In the

second process, the price calculation framework determines (at step 410) whether or not there is an AST representation of the selected price calculation formula already cached. If not, the price calculation framework performs the following steps:

- Step 412: First, the parser (212 – Figure 2) acts to parse the string-based price calculation formula and thereby to create a parse tree representative of the formula;
- Step 414: the price calculation framework (210 – Figure 2) traverses the parse tree representation of the price calculation formula to create an abstract syntax tree (AST) representation for use in subsequent operations, as illustrated in Figure 4A of the drawings. An AST is a tree-wise representation of the grammatical structure of the string-based price calculation formula, as will be familiar to a person skilled in the art;
- Step 416: the price calculation framework (210 – Figure 2) evaluates the AST representation to identify its API dependencies and any formula imports. API dependencies are references to values that are obtained by calling an external function provided by an API (application programming interface), such as a dynamic surcharge that is calculated based on current traffic conditions. Formula imports are other user-defined arithmetic expressions referenced as part of the selected price calculation formula;
- Step 418: if there are any such formula imports, the price calculation framework repeatedly fetches (at step 418) the formula imports and creates the respective AST representations therefor (if not cached), to build a complete AST representation of the selected price calculation formula.

At step 420, the price calculation framework batches and performs calls to external API dependencies concurrently and, at step 422, it populates the results from these computations in a Symbol table. Thus, and as shown in Figure 4B, for the selected price calculation formula, the Symbol Table is populated with values for Coefficient, Coefficient.Basic, Coefficient.Basic.DurationCharges, Coefficient.Basic.DistanceCharges, BasicFare (e.g. operator-defined), Distance (from service provision request) and Duration (also from the service provision request).

It will be appreciated that the Symbol Table may be stored in the memory 126 (Figure 1) or elsewhere, either as part of the communications server 102 or otherwise. In compiler design, a Symbol Table is a data structure created and managed by the compiler to store information about entities such as objects, variable names, and function names. As will be apparent to a person skilled in the art, the Symbol Table may be implemented in a number of different ways, and the present invention is not necessarily limited in this regard. One way of implementing the Symbol Table is with the following recursive type:

```
10 type BasicType ( string | numeral | bool | null )  
type SymbolList[]Object  
type SymbolTable map[string]Object  
type Object ( SymbolTable | SymbolList | BasicType )
```

As will be apparent to a person skilled in the art, in this specific example, the Symbol Table in this implementation is a recursive associative array (i.e., multiple/nested maps), SymbolList is a recursive list, Object is a sum type, and BasicType is a terminating Symbol. The data structure used for the Symbol Table can be implemented with an (in this case, Object) interface that describes the operations that can be performed on it, and define a set of concrete types representing each of the types that an (in this case, Object) can fulfil.

20 The pricing domain language (PDL) described above defines selector expressions, which are combinations of symbols and a selection operator that traverses the Object recursive type. The following is an example of a definition of the "Coefficient" object with its sub-objects, which gives rise to part of the symbol table of Figure 4B:

```
25 {  
  "Coefficient": {  
    "Basic": {  
      "DurationCharges": 10,  
30    "DistanceCharges": 20  
    }  
  }  
}
```

Thus, the pricing domain language allows multiple symbols to have the same name
5 because the user can disambiguate the expression using the Selector expression (e.g.
using the period symbol '.' as in the expression `Coefficient.Basic.DurationCharges`) and
the Symbol Table can store multiple symbols with the same name (as the associative
arrays are nested).

10 It will be evident that the various string-based formulae making up a complete pricing
strategy will include expressions referring to dynamic pricing components such as distance
charges, time charges, dynamic surcharges, dynamic discounts, location fees, demand
charges and many such other components. Many of these dynamic pricing components
will, of course, depend on information provided by other parts of the system. For
15 example, there are may be services that provide the estimated distance and duration of a
specified transport ride as well as geofences of pickup and dropoff locations. The fetching
of such data from sources outside of the price calculation engine 210 is performed by so-
called fetchers. Fetchers in the price calculation engine 210 batch API calls to external
services to reduce round-trip time (see step 420 in Figure 4). Fetchers perform these API
20 calls asynchronously so that price calculations that are not dependent on this data can be
executed concurrently, thereby significantly increasing the speed and efficiency of the
price calculation process. As data is fetched from external sources, it is injected into the
Symbol Table for use in price calculations.

25 As described above, the Symbol Table is used to map a symbol that represents an object in
a price calculation algorithm to the actual object that the symbol represents. As the
functions of a Symbol Table closely resemble a map structure, Symbol Tables are typically
implemented with maps. As briefly discussed above, in this implementation, the Symbol
Table is implemented with object-containing maps, using the Symbol as a key and its
30 corresponding pricing component as a value. As the price calculation expressions are
string-based, it is intuitive to use strings as the data type for keys. Multiple maps are used

as each map corresponds to a single price component. This means that the same Symbol name can be used to refer to different price components if necessary, because the use of multiple maps provides a clean abstraction and clearly indicates which price component is manipulated by a given Symbol. Consider, for example, the expression:

$$\begin{aligned} 5 \quad \text{StandardFare} = & \text{Coefficient.BasicFare} + \text{Route.Distance} * \text{Coefficient.DistanceCharges} + \\ & \text{Route.Duration} * \text{Coefficient.DurationCharges} \end{aligned}$$

It will be apparent that *Coefficient.BasicFare*, *Coefficient.DistanceCharges*, *Coefficient.DurationCharges*, *Route.Distance*, and *Route.Duration* are Symbols in the pricing domain language. BasicFare, DistanceCharges and DurationCharges belong to the Coefficient Pricing Component and are stored in the 'Coefficient' map whereas Distance and Duration are stored in the 'Route' Map (and may e.g. be populated from the service request). For example, referencing BasicFare" in the Coefficient map would yield a numerical value. As explained above, where necessary values in these maps can be populated by Fetchers, e.g. by invoking relevant API calls.

Finally, at step 424, the price calculation framework traverses the AST representation of the selected price calculation formula and uses the values from the Symbol table to calculate the price, by applying the computations (e.g. operations, functions etc) defined by the AST. If a formula import is incorporated in the selected price calculation formula, the result of computation of that operand is also stored in the Symbol table.

Thus, it can be seen that the process implemented by the communications server 102 effectively bypasses or eliminates the compilation and deployment processes required in prior art systems to implement new elements of a pricing strategy. Not only does this negate the need to take the system off-line to add price calculation formulae or make modifications, it also offers advantages in terms of high-velocity development and testing.

The Abstract Syntax Tree (AST) is stored and subsequently accessed and processed using any suitable data representation for representing a tree data structure (e.g. as a set of object nodes, including a root object, with pointers from parent node objects to child

node objects). The AST and the computations stored in the Symbol Table for each selected price calculation formula are cached before they are used to perform the price calculation. They remain cached once the selected price calculation has been performed, which means that the next time that price calculation formula is called, the AST and relevant operand

5 values can be retrieved very quickly, which increases the responsiveness and speed of the system and reduces processing overhead. Furthermore, and for these reasons, a pricing strategy can include many different price calculation formulae to allow for many different respective service provision configurations, without loss of the real-time operation typically required in a service provider system. In an implementation of the

10 communications server 102, a combination of multiple caching layers and a persistent database are used to implement the Formula Database (204 – Figure 2). When read and write operations are performed, the cache and database are read and written in a specific order to enhance performance.

15 Purely by way of example, two layers of cache could be implemented above the persistent database, with the first layer comprising the local instance memory and the second layer comprising a distributed key-value store. When a particular formula is required, Read operations will be performed on the local instance memory layer. If the data is not present, the operation will then be performed on the distributed key-value store layer,

20 followed by the above-mentioned relational database. If the data is found at a particular layer, caches above that layer will be populated accordingly so that subsequent reads would be quicker and more efficient.

Note that Figure 4 illustrates a pull-based caching mechanism (where data such as AST

25 representations and other values are pulled into the cache as needed when not found during lookup e.g. because data was not present or was previously invalidated). As an alternative, the price calculation algorithm could also be implemented with a push-based caching mechanism, as follows:

- 30
1. A user changes one or more coefficient values.
 2. The changes are propagated to the PCE service and the in-memory cache is updated.

3. Whenever the PCE starts all the data is loaded from the database to the cache (including generating and caching ASTs for any formulae in the formula database).
4. The PCE receives a request to calculate the fare for a trip for a particular service type.
5. The PCE fetches the following information from its storage layer concurrently.
 - a. Fetches the set of coefficients from the cache. (already populated by step 3)
 - b. Fetches the formula's AST representation from the cache (already populated by step 3)
6. The PCE fetches the formula's external API dependencies from the cache
 - a. If the list of API dependencies is not in the cache, scan the formula to identify API dependencies and update the cache.
7. The PCE batches and perform calls to external API dependencies concurrently.
8. The results are populated in the Symbol table.
9. The PCE traverses the AST and uses the values from the Symbol table to calculate the fare.

The AST for a formula may be proactively (re-)generated and stored in the cache if a formula is created or changed in the formula database once the PCE is operational, rather than being generated on demand as in the previous example. Alternatively, the approaches could also be combined, e.g. with the cache fully populated at step 3 but then later invalidated and updated on demand as needed if changes occur as in the previously described approach.

In some embodiments, a coefficient store may be implemented as the storage layer that stores static pricing coefficients. PDL allows a user to define access to external dependencies such as APIs and data storage. The coefficient store unifies data storage for static pricing coefficients such as the fare-per-KM-rate (i.e. distance charges), or base fare of a service. The coefficient store acts as the source for populating the symbol table as described above, to provide the values for various symbols encountered in PDL formulae.

The coefficient store implements this storage in a schema-agnostic manner, to allow different applications with different sets of coefficients and features to be supported. The schema of the data stored thus need not be uniform.

5 In an implementation, every set of coefficients is stored in a document, with an associated unique ID. This unique ID is used as a relational key to associate the document with two other data sets:

1. PDL Formula identifier
2. Coefficient Schema Descriptor

10

The formula association is used by the Fetchers to then retrieve the document in tandem with the formula. This document is then converted to an internal Symbol Table representation as described above, which the associated formula can reference. This abstraction allows us to use a formula for multiple applications and services, as long as the
15 formula is compatible with the document schema.

In order to validate the coefficients and PDL formulae that depend on the coefficients, a Schema Descriptor is used to describe the structure of a coefficients document. In an implementation, a single descriptor can be used to describe multiple documents, but a
20 document can only have one descriptor. The descriptor encodes the schema of the document, as well as additional metadata, such as the type of the field, and validations. The schema descriptor is used for three purposes:

1. Validate changes to the coefficient document
2. Validate PDL programs for type correctness
- 25 3. Validate formula associations with coefficient documents

For example, consider the following PDL program:

*Coefficient.BaseFare * 2*

30

This implies that:

1. There is a field with the name "BaseFare" in the coefficients document

2. The type of the “BaseFare” field is a numeric value

As type-checking is a classical programming languages implementation technique, the precise type-checking algorithm will not be described here. This formula will successfully
5 type-check if a schema descriptor which fulfills the two implications is provided. This allows a formula to be associated with a compatible set of schema descriptors.

As formulae can be associated with multiple coefficient sets, the PCE can safely know which formulae can be associated with which coefficient sets by checking if the schema
10 descriptor used to represent the coefficient document is in the compatible set of schema descriptors associated with the formula.

Since the database stores associations between coefficient sets and PDL formulae, the correct coefficient set for a particular formula can be obtained using the stored
15 associations. If multiple coefficient sets are associated with a single formula, then one of those sets may be selected e.g. based on the type of service request (e.g. different coefficient sets could be associated with taxi services and delivery services respectively, thus allowing both service types to reuse the same PDL formula but with a different coefficient set).

20 While the above description refers to certain constant parameter values used in PDL expressions as “coefficients” this should not be interpreted as a limitation on the kind of parameters that can constitute “coefficients”. For example, coefficients need not be limited to multipliers. More generally, the coefficients can be any constant parameter
25 values referenced in the PDL expressions (e.g. additive constants, multipliers, constant parameters to functions such as MIN or MAX etc.) Thus, the coefficient store, and coefficient documents referenced above may hold any type of constant parameter value. Furthermore, the term “constant” is used here merely to distinguish fixed values that generally do not change from one service request to the next (e.g. DistanceCharges
30 specifying price per km) from dynamic parameters that change between service requests (e.g. distance/duration). While thus “constant” with respect to processing of multiple service requests (i.e. they are not service request dependent), these values may

nevertheless change, e.g. being updated from time to time by an operator or automatically.

Figure 5 of the drawings illustrates schematically a complete overview of the price calculation process described above. As previously described, a user-defined arithmetic expression can be created in the editing function 204 and, when called from a formula database (not shown) passes to a lexer and parser 212 to generate a parse tree 502 and, subsequently, an abstract syntax tree (AST) 504, representative of the selected formula.

- 10 In an implementation, the AST may comprise the following types of nodes:
- *Basic Literal (BasicLit)*, which include parameters expressed as integers, floating-point numbers and strings (e.g., 41, 3.14, "Distance", "Duration");
 - *Binary Expression (BinaryExpr)*, which consist of left and right operands and an operator (e.g., 2+3, Distance * DistanceCharges, Duration * DurationCharges);
 - 15 • *Function Calls (FuncCall)*, which represent useful arithmetic functions that cannot be easily represented as mathematical expressions, such as rounding operations or selecting the larger of two numbers (e.g., Max(10, 20), Min(1, 2), RoundUp(10.5));
 - *Module Calls (ModCall)*, which represent the invocation of other price calculation formulae. These nodes provide the system with the capability to use existing price calculation formulae to create more complex ones (as shown in Figure 3, for
- 20 example). This ability to re-use existing price calculation formulae to construct new price calculation formulae significantly reduces unnecessary code duplication, thereby limiting the memory overhead which is particularly significant with large and complex pricing strategies.

25 The AST 504 is used by the price calculation framework 214 (implemented by price calculation engine (PCE) logic), together with values stored in the above-described Symbol Table (not shown in Figure 5). Also illustrated in Figure 5 are the 'EBNF Grammars' 506 and the parser generator 508. These serve to illustrate schematically the 'back' end of the editing function (204 – Figure 2) of the price calculation engine 202. In an implementation

30 of the communications server 102, the grammar used to create the arithmetic price

calculation expressions may be context-free grammar, which can include nested elements, for example:

$$A * B - ((C + D) * E)$$

- In an implementation of the communications server 102, Backus-Naur Form (BNF) may be
- 5 used to define the syntax of the grammar-based language used to enter or modify the arithmetic price calculation expressions in the editing function 204. BNF consists of a series of production rules whereby each rule has a name and a body. The body defines how to decompose a rule into a series of terminals or rules.
- 10 There are a number of different methods of converting grammars into parsing algorithms, as will be apparent to a person skilled in the art. In an implementation of the communications server 102, a parser generator 508 is used. Parser generators are mature and well-researched tools that use grammars as inputs to generate parsers in various programming languages. These generated parsers can process complicated languages
- 15 efficiently, as will be known to a person skilled in the art. Some well-known examples of open-source parser generators include ANTLR, Bison and Yacc. Different parser generators typically accept different types of grammar notations as inputs. These generated parsers may also employ parsing algorithms. In an implementation of the communications server 102, ANTLR may be used, which accepts grammars in Extended
- 20 Backus-Naur Form (EBNF) notation. The EBNF grammars of an implementation may be written in a recursive manner as it describes precedence levels and nested expressions in an intuitive and understandable manner.
- As described above, the lexer and parser 212 generated from the EBNF grammars 506
- 25 parses a string-based price calculation formula into a parse tree. The price calculation engine 202 converts the parse tree into an abstract syntax tree by removing unnecessary information (e.g. syntax tokens not needed to express the meaning of the parsed code such as bracket or whitespace tokens). Note that in some implementations, the parser may generate the AST directly without first creating an explicit parse tree representation.
- 30 The calculation framework 214 then traverses the AST 504 and calculates the price using the operations defined in the AST and the values stored in the Symbol Table.

- Some implementations envisage extending a single pricing strategy across two or more different service types. Thus, for example, a single pricing strategy can be generated to cover two or more different transport services (e.g., taxi, car share and delivery). Price calculation expressions can be entered into the same formula database to cover all of the
- 5 service provisions, because each formula can be paired with different sets of coefficients to cater for different use cases. Users can configure formulae and coefficients via a single user interface. In order to access the correct price calculation for a particular service provisions request, the following information may be required in an embodiment:
- Pickup and dropoff location
 - 10 • Time of journey
 - The index of the requested service type
 - The name of the formula to be used
 - The index of the set of coefficients to be used
- 15 The PCE 202 receives a request to calculate the fare for a trip for a particular service type. The PCE first checks the cache layer to determine if the set of coefficients and/or the AST representation for the required formula are cached. If the coefficients are not cached, the PCE 202 fetches them from a database and updates the cache (so they are there for next time the price calculation is called). If the AST representation is not cached, the PCE
- 20 fetches the string-based arithmetic expression defining the price calculation, parses the expression to create a parse tree, converts the parse tree into an AST and updates the cache with the AST representation. Once the set of coefficients and AST representation are obtained, the PCE checks the cache for the price calculation s API dependencies and if they are cached, fetches them. If the API dependencies are not cached, the PCE scans the
- 25 formula to identify API dependencies and updates the cache accordingly. Next, the PCE batches and performs calls to the external API dependencies concurrently and, finally, populates the results in the Symbol Table before using the AST representation and the values in the Symbol Table to calculate the price.
- 30 The described system allows users to create and maintain large sets of PDL formulae to address different services, service types and circumstances. Instead of complex

conditional code that adapts price computations to particular circumstances, the user can simply create different PDL formulae and add these to the database. As an example, the following PDL formulae relate to fare calculations with and without a basic fare component:

5
$$F1 = \text{BasicFare} + \text{Distance} * \text{DistanceCharges} + \text{Duration} * \text{DurationCharges}$$
$$F2 = \text{Distance} * \text{DistanceCharges} + \text{Duration} * \text{DurationCharges}$$

Whether BasicFare is required or not are represented as two different equations. A vehicle that requires BasicFare to be included will use F1, whereas a vehicle that does not will use
10 F2. The formula to use is specified in the service request / price calculation request sent to the PCE. As a result, there is no need to implement conditional logic since the choice of whether to include the basic fare component was made at the configuration stage.

Exemplary embodiments thus provide a communications server apparatus for processing
15 a service provision request for a service. The communications server apparatus comprises a processor, a memory and a formula database configured to contain a set of string-based arithmetic expressions, each expression defining a price calculation to be applied to a service provision request. The apparatus is configured, under control of the processor, to execute instructions stored in the memory to perform the following steps. Data
20 representative of a specified service provision request is received including data representative of one or more parameters associated therewith, wherein a price for said specified service provision is at least partially dependent on the values of said parameters. In response to receipt of said data representative of said specified service provision request, a price calculation to be performed for the service provision request is identified,
25 the price calculation associated with a given one of the set of string-based arithmetic expressions; an abstract syntax tree, AST, corresponding to the given string-based arithmetic expression is obtained, the AST having been generated from the given string-based arithmetic expression by parsing the expression; a Symbol Table is populated with values for said parameters associated with said specified service provision request; and a
30 price for said specified service provision request is calculated using said AST and the values in said Symbol Table. Exemplary embodiments also provide a method for performing the above steps.

While described mainly in relation to on-demand transportation services, the pricing domain language (PDL) can be used with the PCE to define and implement pricing formulae for a wide range of products, services, and business offerings. As mentioned
5 above, these may be products and/or services for distinct sectors or categories, including but not limited to, logistics (e.g., delivery of documents, packages, parcels, goods, etc.), transport (e.g., ride-hailing services, ride-sharing services, etc.), finance (e.g., payment, banking, insurance, etc.), food (e.g., food ordering, food delivery, etc.), goods services (e.g., purchase and/or delivery of groceries, packaged goods/foods, household products,
10 etc.), trade or handyman-related services, etc. Some particular examples are given below.

Examples of PDL Formulae for E-Commerce / Online stores

Pricing strategies for e-commerce sites and online stores are relatively straightforward.
15 They generally include the prices of items in the shopping cart, less applicable discounts from promotions and campaigns, plus applicable charges such as shipping or delivery fees.

The following PDL formulae are examples of pricing strategies for e-commerce services.

20 $TotalPriceOfItems = PriceOfItems - Discounts$

$DeliveryFee = 5$

$Tax = TotalPriceOfItems * CountrySpecificPercentage$

25

$FinalPrice = TotalPriceOfItems + DeliveryFee + Tax$

Examples of PDL Formulae for E-Commerce / Online stores with On-Demand Delivery

30 On-Demand Delivery for e-commerce services operate with similar pricing strategies as e-commerce services described in the previous section. However, these pricing strategies

are more “dynamic” in nature as they may incorporate dynamic components such as “Demand Charges”, and “Location Fees”, which takes traffic conditions, passenger demand, driver supply, and other real-time factors into account when pricing a delivery.

- 5 The following PDL formulae provide examples of pricing strategies for e-commerce services with On-Demand delivery.

$$\text{TotalPriceOfItems} = \text{PriceOfItems} - \text{Discounts}$$

10
$$\text{DeliveryFee} = \text{BasicFare} + \text{Distance} * \text{DistanceCharges} * \text{Max}(\text{DemandCharges}, \text{MinDemandCharges}) + \text{LocationFees}$$

$$\text{Tax} = \text{TotalPriceOfItems} * \text{CountrySpecificPercentage}$$

15
$$\text{FinalPrice} = \text{TotalPriceOfItems} + \text{DeliveryFee} + \text{Tax}$$

Examples of PDL Formulae for Restaurant Dine-in or Takeaway

- Pricing strategies for dine-in restaurants generally include the total price of food orders, plus applicable charges such as service charges, goods and services taxes, and credit card surcharges. The following PDL formulae give examples of pricing strategies for restaurant dine-in and takeaways.
- 20

$$\text{GoodsAndServicesTax} = 8\%$$

25

$$\text{PriceAfterTaxes} = \text{TotalPriceOfFood} + (\text{TotalPriceOfFood} * \text{GoodsAndServicesTax})$$

$$\text{ServiceCharge} = \text{PriceAfterTaxes} * \text{RestaurantSpecificPercentage}$$

30
$$\text{CreditCardSurcharge} = \text{PriceAfterTaxes} * 5\%$$

$$\text{FinalPrice} = \text{PriceAfterTaxes} + \text{ServiceCharge} + \text{CreditCardSurcharge}$$

Examples of PDL Formulae for On-Demand Food delivery

On-Demand Food Delivery services operate with similar pricing strategies as restaurant
5 dine-in or takeaway described in the previous section. However, these pricing strategies
are more “dynamic” in nature as they may incorporate dynamic components such as
“Demand Charges”, and “Location Fees”, which takes traffic conditions, passenger
demand, driver supply, and other real-time factors into account when pricing a delivery.
The following PDL formulae are examples of pricing strategies for On-Demand Food
10 Delivery.

$$\text{DeliveryFee} = \text{BasicFare} + \text{Distance} * \text{DistanceCharges} * \text{Max}(\text{DemandCharges}, \text{MinDemandCharges}) + \text{LocationFees}$$

15
$$\text{FinalPrice} = \text{TotalPriceOfFood} + \text{DeliveryFee} - \text{Discounts}$$

It will be appreciated that the invention has been described by way of example only.
Various modifications may be made to the techniques described herein without departing
from the spirit and scope of the appended claims. The disclosed techniques comprise
20 techniques which may be provided in a stand-alone manner, or in combination with one
another. Therefore, features described with respect to one technique may also be
presented in combination with another technique.

Claims

1. A communications server apparatus for processing a service provision request for a service, the communications server apparatus comprising a processor, a memory and a formula database configured to contain a set of string-based arithmetic expressions, each
5 expression defining a price calculation to be applied to a service provision request, and being configured, under control of the processor, to execute instructions stored in the memory to:

receive data representative of a specified service provision request including data representative of one or more parameters associated therewith, wherein a price for said
10 specified service provision is at least partially dependent on the values of said parameters; and, in response to receipt of said data representative of said specified service provision request:

identify a price calculation to be performed for the service provision request, the price calculation associated with a given one of the set of string-based arithmetic
15 expressions;

obtain an abstract syntax tree, AST, corresponding to the given string-based arithmetic expression, the AST having been generated from the given string-based arithmetic expression by parsing the expression;

populate a Symbol Table with values for said parameters associated with said
20 specified service provision request; and

calculate a price for said specified service provision request using said AST and the values in said Symbol Table.

2. A communications server apparatus according to claim 1, configured, when
25 obtaining the AST, to:

determine if an AST representative of the given string-based arithmetic expression is saved in a cache associated with said communications server apparatus, and, if so, retrieve said AST from said cache; if not:

retrieve the given string-based arithmetic expression from said formula
30 database;

parse the given string-based arithmetic expression to create an AST corresponding to the expression; and

save said AST to said cache.

3. A communications server apparatus according to claim 2, configured to invalidate
cache data storing an AST representation for a particular string-based arithmetic
5 expression in the cache in response to a change to the particular string-based arithmetic
expression in the formula database.
4. A communications server apparatus according to claim 1, configured to parse the
given string-based arithmetic expression, generate the AST from the string-based
10 arithmetic expression and store the AST in the cache prior to receiving the service request
and/or in response to addition or modification of the string-based expression in the
formula database.
5. A communications server apparatus according to any of the preceding claims,
15 further configured to traverse said AST and perform any calls to external APIs referenced
therein and to populate the Symbol Table with values obtained from said external APIs.
6. A communications server apparatus according to claim 5, further configured to
store said values obtained from said external APIs in said cache.
20
7. A communications server apparatus according to any of the preceding claims,
comprising a user interface configured to allow a user to enter new string-based
arithmetic expressions and/or modify existing string-based arithmetic expressions, and to
save new and/or modified arithmetic expressions to said formula database.
25
8. A communications server apparatus according to claim 7, wherein said user
interface is configured to allow a user to create a new arithmetic expression using one or
more pre-existing arithmetic expressions saved in said formula database as one or more
respective operands of the new arithmetic expression, and save said new arithmetic
30 expression to said formula database.

9. A communications server apparatus according to any of the preceding claims, wherein at least some of the string-based arithmetic expressions include at least one constant parameter having a predetermined value, and the communications server apparatus being further configured to determine if any constant parameter values in the
5 given arithmetic expression are saved in the cache and, if so, retrieve said constant parameter values from said cache and incorporate them into said Symbol Table.
10. A communications server apparatus according to claim 9, further configured to, if said constant parameter values are not stored in said cache, retrieve said constant
10 parameter values from a database associated with said communications server apparatus, incorporate them into said Symbol Table, and update said cache with said constant parameter values.
11. A communications server apparatus according to any of the preceding claims,
15 configured to store a plurality of sets of constant parameter values in a database, to select a given one of the sets of constant parameter values based on the service request and/or the given string-based arithmetic expression, and to obtain one or more values of constant parameters referenced in the given string-based arithmetic expression from the selected set and incorporate the values into the Symbol Table for use in processing the
20 AST for the service request.
12. A communications server apparatus according to claim 11, wherein the database is configured to store an association between the string-based arithmetic expression and one or more constant parameter value sets, the apparatus configured to select the
25 constant parameter value set based on the stored association.
13. A communications server apparatus according to claim 11 or 12, configured to store each constant parameter value set as a document, associate a schema descriptor with each document encoding a schema of the document, and use the schema descriptor
30 for a given document to perform at least one of: validating one or more changes to the document, validating a string-based arithmetic expression associated with the document

for type correctness; and validating an association between a string-based arithmetic expression and the document.

14. A communications server apparatus according to any of the preceding claims,
5 wherein the symbol table is arranged to map one or more symbols referenced in the string-based expression and/or AST to values associated with the symbols, the apparatus configured, when performing the price calculation, to traverse the AST, lookup values for symbols referenced in the AST in the symbol table and use the values to perform the calculation defined by the AST.
- 10 15. A communications server apparatus according to any of the preceding claims, wherein said string-based arithmetic expressions are written in a domain language specified using a context-free grammar.
- 15 16. A communications server apparatus according to claim 15, wherein said string-based arithmetic expressions are written in a domain language specified using an Extended Backus-Naur Form, EBNF, grammar.
17. A communications server according to any of the preceding claims, further
20 comprising a parser generator for generating a parser for parsing said identified arithmetic expression to create a parse tree, the apparatus configured to create the AST from said parse tree.
18. A communications system for implementing a service comprising a user
25 communications device for generating a request for a specified service provision, and a communications server apparatus according to any of the preceding claims configured to receive, from said user communications device, data representative of said specified service provision request including data representative of one or more parameters associated therewith, wherein a price for said specified service provision is at least
30 partially dependent on the values of said parameters.

19. A communications system according to claim 18, wherein said communications server apparatus further comprises a user interface configured to allow a user to enter new string-based arithmetic expressions and/or modify existing string-based arithmetic expressions, and to save new and/or modified arithmetic expressions to said formula database.

20. A communications system according to claim 19, wherein said user interface is configured to allow a user to create a new arithmetic expression using one or more pre-existing arithmetic expressions saved in said formula database as one or more respective operands of the new arithmetic expression, and save said new arithmetic expression to said formula database.

21. A method, performed in a service provider apparatus, for processing a service provision request for a service, the service provider apparatus comprising a formula database containing a set of string-based arithmetic expressions, each expression defining a price calculation to be applied to a service provision request, the method comprising:

receiving data representative of a specified service provision request including data representative of one or more parameters associated therewith, wherein a price for said specified service provision is at least partially dependent on the values of said parameters; and, in response to receipt of said data representative of said specified service provision request:

identifying a price calculation to be performed for the service provision request, the price calculation associated with a given one of the set of string-based arithmetic expressions;

obtaining an abstract syntax tree, AST, corresponding to the given string-based arithmetic expression, the AST having been generated from the given string-based arithmetic expression by parsing the expression;

populating a Symbol Table with values for said parameters associated with said specified service provision request; and

calculating a price for said specified service provision request using said AST and the values in said Symbol Table.

22. A computer program product comprising instructions for implementing the method of claim 21.
23. A computer program comprising instructions for implementing the method of
5 claim 21.
24. A non-transitory storage medium storing instructions which, when executed by a processor, cause the processor to perform the method of claim 21.