A DSL For Logistics Clouds

Bill Karakostas
School of Informatics, City University London
Northampton Square
London, UK
billk@soi.city.ac.uk

Takis Katsoulakos
Inlecom Ltd
Knowledge Dock Business Centre, 4-6 University Way
London, UK
takis@inlecom.com

Abstract— **Cloud is a new area of specialization in the computing world, and, as such, it has not been explicitly addressed by traditional programming languages and environments. Therefore, there is a need to create Domain Specific Languages (DSLs) for it. This paper presents such a DSL that targets logistics clouds, i.e. networked resources and systems of logistics organisations. The DSL is implemented on top of the functional concurrent language Erlang and its distributed data management system Mnesia. The paper presents features of the DSL that implement commonly occurring use cases in the logistics cloud such as message exchange, document sharing and notifications. We show how program features in this DSL map to the underlying Erlang/OTP runtime.**

Keywords- DSL; Logistics Cloud; Erlang/OTP; Mnesia; transport logistics; functional programming

I. INTRODUCTION

Community clouds are implementations of Clouds by a community of organisations such as logistics companies that agree to virtualise and share their computing resources. In contrast to a generic, "horizontal cloud", components of a logistics cloud are custom tailored to the specific needs of the logistics application area [7].

Effectively, a logistics cloud is a networked data and computing infrastructure that virtualises resources (documents, data, systems and applications) for a logistics business network, to which nodes can dynamically be added and removed. Physical resources in logistics (such as cargo) are, by nature, mobile, and are handled and monitored by multiple IT systems. For cooperative processes, it is therefore important that the information about the state of logistics resources remains independent from location and physical formats of the systems that handle it. Resources and operations on them must therefore be abstracted in an implementation independent form, following the principles of Representation State Transfer (REST) [6]. This allows the participants of the logistics cloud to perform collaborative processes without concern about the physical format and location of data and applications, i.e. to work in a Cloud environment. According to the iCargo project [9], such a Cloud is a 'parallel universe' mirroring logistics processes, resources and data, and offering capabilities for co-operative synchronized and real-time management of transport resources (i.e. intelligent planning and controlling transport logistics chains) to optimise efficiency, quality and environmental performance. The paper presents a Domain Specific Language (DSL) for developing cloud applications for logistics organizations.

The rest of the paper is structured as follows: Section 2 overviews Cloud DSLs and explains the rationale and design objectives for the proposed DSL. Section 3 introduces the main architectural concepts of the logistics cloud, while Section 4 presents the main features of the language. Section 5 highlights the main use cases for the DSL as investigated in the iCargo project. The last section highlights the plans for further research and development.

II. DOMAIN SPECIFIC LANGUAGES AND CLOUDS

A DSL is a programming language or an executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain [3]. DSLs have been used in many domains, particularly due to their expressiveness, runtime efficiency and reliability due to their narrow focus. More recently, DSLs for clouds have been proposed for high performance computing [2] business process management [1] and business applications [8]. Data cloud specific DSLs, such as Pig Latin from the Apache Pig project, are employed for analyzing large data sets [10].

Currently, logistics applications are implemented in general purpose languages (GPL) such as Java and C# and Web languages such as Javascript and HTML. Message exchanges are typically implemented in XML, while system interfaces are specified as Web services. However, logistics organisations and chains have become increasingly distributed and virtualised. Current development technologies fall short in realising the full potential of RESTful architectures and of the Cloud. The aim of our research has been to exploit the potential of concurrent functional languages such as Erlang [5] and distributed data management systems such as Mnesia [5] in developing logistics applications that take advantage of the Cloud's potential. The use of Erlang to develop RESTful applications has been proposed before by S. Vinoski [13], and the potential of functional languages on the Cloud has also been advocated by J. Epstein *et al* [4]. However, the learning curve for such technologies can be steep. A DSL could help towards easing the adoption of functional concurrent languages, while maintaining their expressiveness and power in developing business applications for the Cloud.

 ISBN: 978-1-61208-271-4

A. Rationale for the Design of the DSL

One of the design goals was to preserve the benefits of Erlang such as the built-in, actor based, concurrency model, while easing the learning curve for the typical logistics application developer. Erlang programming has limitations such as the unconventional syntax, the lack of types, and the general lack of familiarity with functional programming styles amongst developers.

At the same time, the design of the DSL had to address an easier to read and understand syntax (i.e. by avoiding the excessive use of parentheses and brackets) and support for types. To avoid designing yet another GPL, however, only predefined types, derived from a Common Reference Model (Framework) for logistics domain were allowed. The Common Framework used was developed in EU projects such as e-Freight and iCargo [9] and provided the basis for the main domain concepts of the logistics DSL.

III. MAIN CONCEPTS

Erlang is a functional programming language used to build massively scalable soft real-time systems [5]. A distributed Erlang system consists of a number of nodes (Erlang runtime systems) communicating with each other. A node is an executing Erlang runtime system which has been given a name. Each such runtime system is called a node. The distribution mechanism is implemented using TCP/IP sockets. Mnesia is a multiuser distributed data management system written in Erlang, which is also the intended target language. In our prototype implementation, the execution of a program written in the DSL results in several spawned Erlang processes. These processes communicate with other processes across the logistics cloud, and manipulate Mnesia tables holding information about logistics resources. In a logistics cloud, the physical implementation and address of resources is virtualised. Resources are identified using logical Uniform resource identifiers (URIs) constructed from domain names of their owners and literals such as internal identifiers. Our approach assumes that logistics cloud participants have unique URIs (i.e. domain names in the Domain Name System) and all other resources acquire their unique identifiers relatively to the URIs of their owners. This avoids the need to assume (and agree upon) resource identifiers that are globally unique across the whole logistics Cloud.

We implement RESTful (PUT and GET) operations in our approach, but with functional semantics to maintain consistency with the Erlang underpinnings.

The code written in the DSL is translated with the use of a pre-processor (similar to Erlang's pre-processor) to Erlang modules that can be loaded and executed by the Erlang emulator. A program in the DSL is therefore an Erlang module containing function definitions that can be compiled and executed by the Erlang emulator. A typical execution spawns several Erlang processes. These can run on different nodes of the logistics cloud. As with standard Erlang, inter-process communication is via message exchanges.

IV. GENERAL SYNTACTIC CONVENTIONS

To reduce the learning curve, the DSL has a minimal set of constructs and relies on predefined domain types that are manipulated in a RESTful way to create and access resources. To distinguish between the DSL and regular Erlang language constructs, the former must begin with an underscore and consist of all capitals letters. Tokens that are not recognised by the pre-processor as reserved must be valid Erlang terms.

Reserved keywords fall under the categories of:

- Logistic Roles e.g. *_CONSIGNER, _FREIGHTFORWARDER, _CONSGINEE*
- Resource Types: Business documents, e.g.: *_TEP* (transport execution plan exchanged between logistics partners), administrative forms, etc. notification types such as *_DISPATCH_NOTICE*
- Resource read and modify operations using *_NEW* and _GET commands.
- Control Flows such as *_FOREACH* for iteratively applying a function to the members of a list
- Some Erlang data types such as lists constructors ('[]') and operators such as ! (for sending messages to processes) are also explicitly supported by the DSL.

Logistics roles are implemented as Mnesia transactional queries, while business document types are implemented as Mnesia tables and document instances as document records. This is further explained in the following section.

V. USE CASES

Below we show some typical use cases for this DSL, highlighting the syntax of the commands, the effect of the operations and an explanation of their underlying Erlang/Mnesia semantics.

A. Defining users and roles in the Logistic Cloud

Each organisations participating in the logistics cloud implements a (distributed) Erlang network node, for example the following set of nodes that correspond to 4 participating logistics organizations is defined in a logistics cloud:

consigner1@org1.com,consignee1@org2.com
freightforwarder1@org3.com, carrier1@org4.com

The above logistic cloud participants, agree, for example, to share data between them. A participant, such as freightforwarder1, may know all other participants (due to its coordinating role), and can therefore, initiate the sharing of the Mnesia database by executing the following command locally:

mnesia:create_schema([consigner1@org1.com, consignee1@org2.com, freightforwarder1@org3.com, carrier1@org4.com]

More participants can be added to the logistics cloud at any point, dynamically, by following this approach. Mnesia tables are automatically created for each supported resource

type on every participant node, subject to sharing declarations (explained below). A Mnesia table is a collection (more precisely, a bag) of records. Records (instances of resources) are created by participants as explained below.

B. Sharing resources amongst participants

The general syntax for explicitly sharing resources (tables) with other cloud participants is:

<Resource type> _SHARE_WITH <list of participants> _AS <Qualifications>

This results in changes to the corresponding table replication properties in the underlying Mnesia database, so that the table can be shared as read-only, read-write, and so on.

C. Creating new resources

The general syntax for creating records (instances of resources) is:

_NEW <resource type> _WITH <key-value list>

For example, to create a new arrival notice, the following command is used:

_NEW _ARRIVAL_NOTICE _WITH {ref="12345", status= "OK"}

The result of the operation is to add a new arrival notice record to the Mnesia table (bag) *arrival_notice*.

The internal record definition in Mnesia is *record(arrival_notice, {ref :: string(), status :: string()}).*

Note that *_NEW* does not have to specify the location of the target database, as the record is added to the local table of the node where the command is executed and replicated according to the policies defined for that table.

D. Querying Resources

The general syntax for retrieving resource records is: *_GET <Resource Type> _WITH <Qualifications>*

This returns a list of instances (records) of type 'Resource Type' that match 'Qualifications'. If the Qualifications part is omitted, all records (up to a maximum system imposed limit) are returned. This list of records can then be accessed using the _FOREACH operator.

Qualifications are logical expressions that specify range and other logical conditions on the properties of resources being queried.

For example, to retrieve all consignments for consigner with id *consigner1@org1.com* that have status *'dispatched'*, the following query can be used*:*

_GET _CONSIGNMENT _WITH {consigner = "consigner1@org1.com", status = "dispatched"}

Internally, the pre-processor converts queries like the above to Erlang 'list comprehension' style of queries that are then executed as Mnesia transactions.

E. Messaging

Messaging has been inspired by REST messaging approaches such as RESTMS [11].

The general syntax for messaging is

<Recipient List> ! _MESSAGE_TYPE _WITH {key value list} _AS <Message Format>

Where 'Recipient List' can be the result of a query that returns the identifiers of recipients. The following code for example, sends a message (formatted as XML) containing a dispatch notice, to the owner (consigner) and the recipient (consignee) of a consignment:

[consigner1@org1.com, consignee1@org2.com] ! _NEW _DISPATCH_NOTICE _WITH {ref= "12345", status= "dispatched"} _AS _XML

Additional parameters can be specified, for example, regarding the exact time the message is to be sent, how to handle errors such as no replies (timeout conditions) and so on. Internally, this is converted into message sending operations to the message listening processes of the recipient nodes. Such processes are automatically spawned when the nodes join the logistics Cloud. Messages can also be sent to recipients outside the logistics cloud by using call-back methods.

F. Event Notifications

Logistics cloud participants can publish and subscribe to events in the logistics cloud. This is often a more flexible approach than direct messaging as it decouples the senders and consumers of event notifications.

A Consigner *consigner1@org1.com*, for example, can subscribe to notifications when dispatch notices are created. The general syntax for subscriptions is:

_SUBSCRIBE_TO <Resource Type> _WITH <Conditions>

Internally, this is implemented by an Erlang process on node *consigner1@org1.com* that subscribes to update events on table *dispatch_notice*, using the command *mnesia:subscribe({table, dispatch_notice, simple}).*

If the monitoring process receives a message notification such as *{write, NewRecord, ActivityId},* it will check that the conditions are satisfied, and if they are, the process will notify the callback process on the *consigner1@org1.com* node.

VI. CONCLUSIONS AND FUTURE WORK

Functional concurrent languages have a great potential for building the next generation of Cloud applications, due to scalability, side effect free code and ease of transformation to multiple representation formats (XML, JSON,…) of Cloud resources. Our approach is at the early stages of developing an easy to use Cloud development environment for logistics applications. We are currently investigating security features (authentication, authorization at organization and user role level) for the proposed DSL, and also support for transactional rollback and error handling both at the pre-processing stage and at runtime. We also plan to explore alternative target Cloud environments that support functional programming languages, such as Scala. After we complete the development of the pre-processor, we plan to develop a full blown transport logistic Cloud based collaborative application within the iCargo project. This application will demonstrate an implementation of the Common Framework in the DSL and the use of associated interfaces to facilitate the connection of logistics companies to the iCargo ecosystem.

 ISBN: 978-1-61208-271-4

ACKNOWLEDGMENT

The iCargo project "iCargo - Intelligent Cargo in Efficient and Sustainable Global Logistics Operations" is co-funded by the EU FP7 Program.

REFERENCES

[1] B. Karlis, C. Grasmanis, A. Kalnins, S. Kozlovics, L. Lace, R. Liepins, E. Rencis, A. Sprogis, and A. Zarins. "Domain Specific Languages for Business Process Management: a Case Study". Proc. the 9th OOPSLA Workshop on Domain-Specific Modeling. 25-26 October 2009.

[2] C. Bunch, N. Chohan, and K. Shams. "Neptune: A Domain Specific Language for Deploying HPC Software on Cloud Platforms" UCSB Technical Report #2011-02 .

[3] A. van Deursen, P. Klint, and J. Visser. "Domain-specific languages: an annotated bibliography". SIGPLAN Not., 35(6):26--36, 2000.

[4] J. Epstein, AP Black, and S. Peyton-Jones. "Towards Haskell in the cloud." Proc. The 4th ACM symposium on Haskell, ser. Haskell. ACM, New York, NY, USA, pp. 118–129.

[5] Ericsson AB. "Erlang/OTP System Documentation 5.8.3" March 14 2011.

[6] R. Fielding. "Architectural Styles and the Design of Network-based Software Architectures". Ph.D. Thesis. University of California, Irvine, 2000.

[7] B. Holtkamp, S. Steinbuss, H. Gsell, T. Loeffeler, and U. Springer. "Towards a Logistics Cloud" Proc. 2010 Sixth International Conference on Semantics, Knowledge and Grids Beijing, China November 2010.

[8] M. Kumar. "Domain Specific Language Based Approach for Developing Complex Cloud Computing Applications." Masters thesis. Wright State University 2011.

[9] J. Pedersen. "Frameworks and Applications for Logistics". Proc, 2nd European Conference on ICT for Transport Logistics (ECITL), Venice 2009,

[10] Pig Latin Basics. Available from http://pig.apache.org/docs/r0.10.0/basic.html [retrieved March 2013]

[11] RESTMS. Available from http://www.restms.org/ [retrieved March 2013]

[12] D. Stieger, M. Farwick, B. Agreiter, and W. Messner. "DSLs to fully generate Business Applications". Available from www.jetbrains.com/mps/docs/MPSShowcase.pdf [retrieved March 2013]

[13] S. Vinoski. "RESTful Services with Erlang and Yaws". InfoQ, March 31, 2008.

 ISBN: 978-1-61208-271-4