**Motivation**

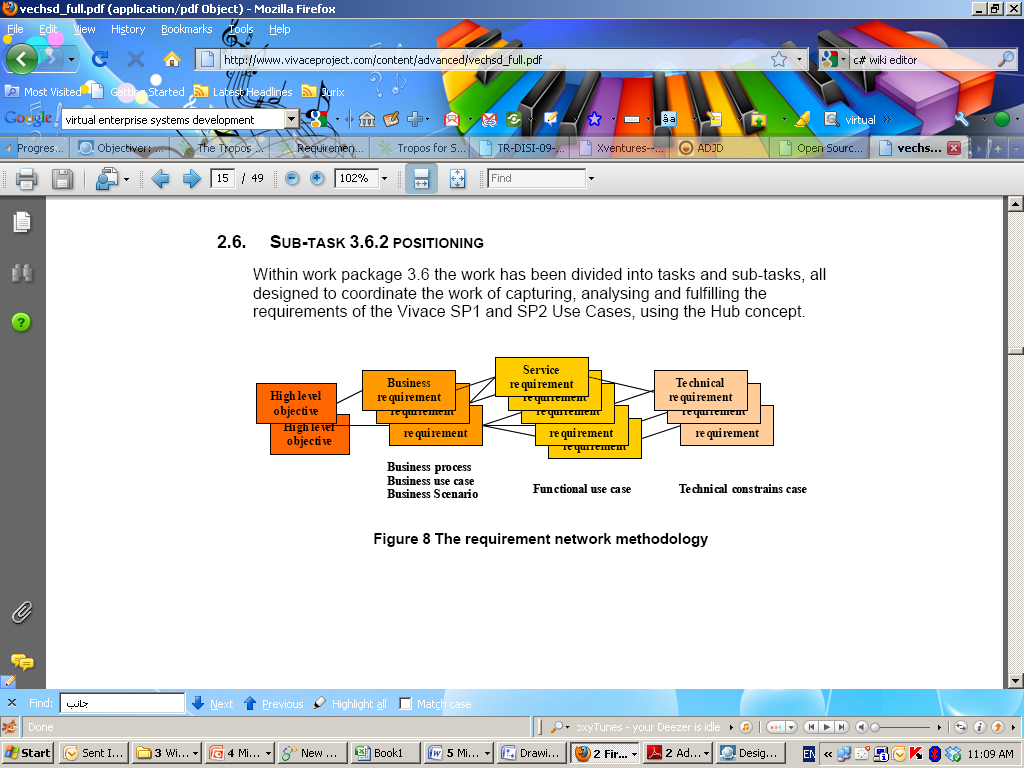
If a process view becomes centric to any economic activity, then the structure of the resources required supporting it, including by which they are owned, and where they are located becomes just an optimization decision on the proper means to support a process. Here comes the concept of a virtual enterprise.

We will first present the ultimate definition of a virtual enterprise. Then we will propose our view of the succession of steps such an enterprise creation implies including:

* *How to design processes*
* *How to map processes on resources required to support them;*
* *How to contract resources from other companies when appropriate;*
* *How to operate the virtual organization; and finally*
* *How to keep control on such complex organizations.*

The Internet offers any individual the ability to exchange information with anyone in the world as if they were in the same village—the “global village.” This makes it possible to create communities of people with a common interest where distance is irrelevant. Electronic commerce is part of the scenario because any electronic shop is potentially part of the village. Similarly, Workflow can spread across company boundaries and anywhere in the world on the Internet. This opens up a new way of doing business through homework, mobile work and virtual enterprises. It is possible to realise cooperation between economic actors for common undertakings and allow dynamic reconfiguration as the need arises. In the long term this will bring a major breakthrough in productivity, organization, international exchanges and economic growth through:

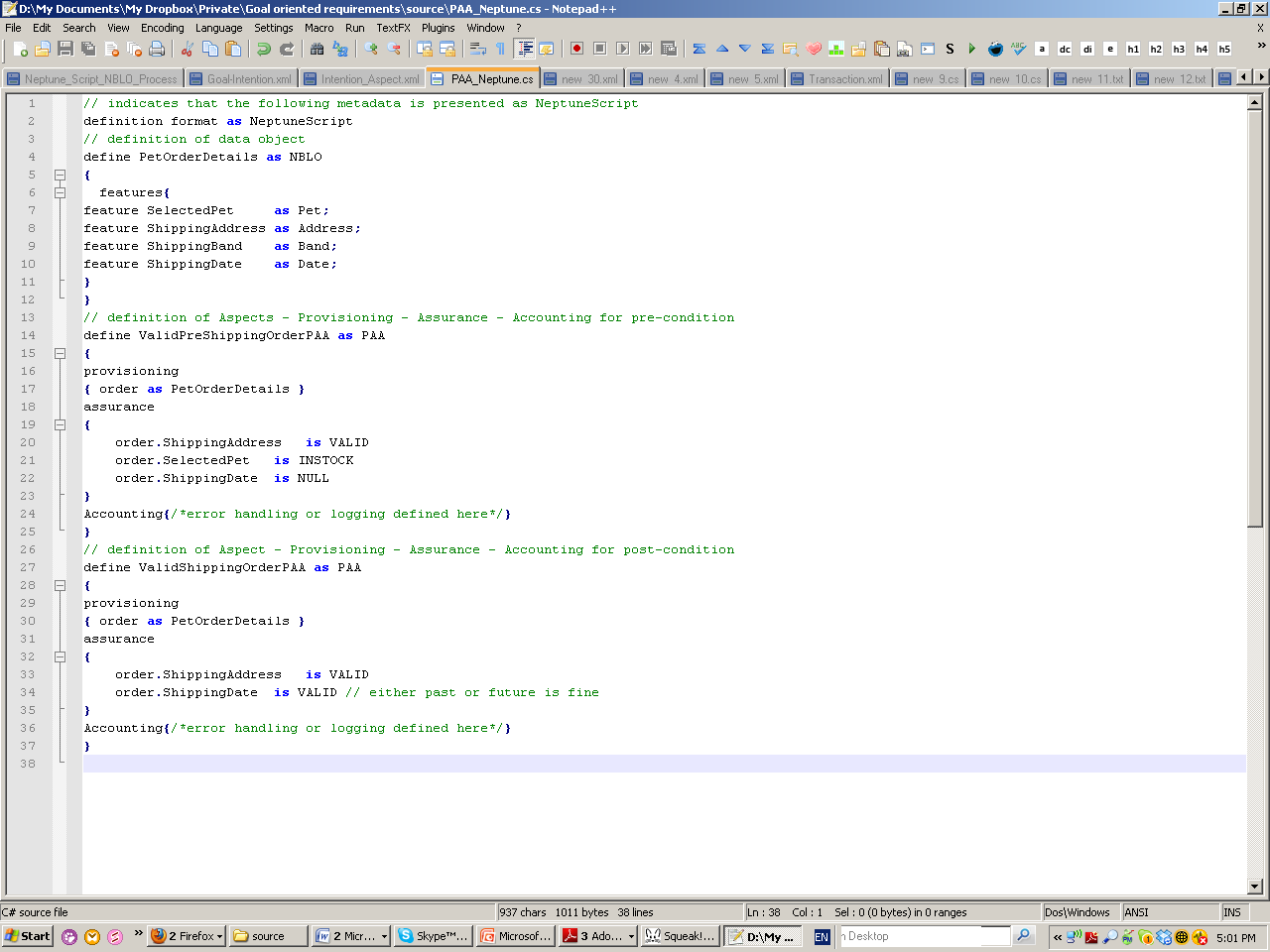
* Innovative products and services combining communication, electronic commerce and
* business process automation to provide effective and low cost customer service worldwide
* Innovative processes where the client is a direct participant in the process and
* Workflow-enabled processes track transactions across unit, company and enterprise
* boundaries
* Innovative organizations relying on Workflow-based end-to-end business processes to
* provide the best possible service, in spite of the internal structure being adapted to
* match market needs
* Dynamic win-win cooperation between enterprises and individuals. Better service for
* the market will be based upon optimum performance of each of the assembled
* Partners being enabled through Workflow-controlled business processes.



**Introduction**

The whole core idea behind Neptune is a methodology to scout for the appreciation of semantics in codebase or in model, or in another mean exploring using AOP and abstracting aspects from a model or from a codebase, then feature these aspects using descriptive declaration with findings and requirements, both findings and requirements should be proposed by developer first at design time, when findings are what contextually exist from what the code intends to do in terms of Aspects, while requirements are what does not exist or what does these Aspects “Need” to achieve foreseen Goal of code or model. This basic formation of the idea behind Neptune is translated into “language support” with logic to help specifying how will Aspects behave to reach the Goal. The techniques that Neptune uses in the notion of “reflection” where any language (syntax and logic) can be passed into several stages before being compiled instead of direct parsing, implementation/execution. These stages can be a parsing of the code syntax, then exemplifying/distilling some expressions from tokens then following certain interpretation rules in defined grammar to interpret the codebase syntax into metadata code and objects. These objects will do any further logic as desired by a separate object model for any desired purposes like in my case Goal agent. During compile time (runtime) the challenge is arise is that Intention modeling is done through one party (intention modeler) while during compilation process this developer finds realistically that he needs to add further support to other Aspects as Neptune is eager to distill more Aspects through appreciation of semantics of the code. This dynamic interaction of whom to provide this “language support” is quite necessary, especially when realizing the major role in this role based modeling. Quite several parties are involved in providing their support from different aspect views. But since the basic intuitive goal of modeling is to perform runtime acquisition of a software application, and since the target desired state of the system is to perform preventative actions against any change in its organics; there will be only one initiator of this model and all relevant actors whether partners, influencers,..etc should contrive all means, plans to support actuating his intentions into achieved set of Goals. The next challenge of course is how to build a level of trust by balancing between intended behaviors imposed by an intention modeler and Value streams to all parties.

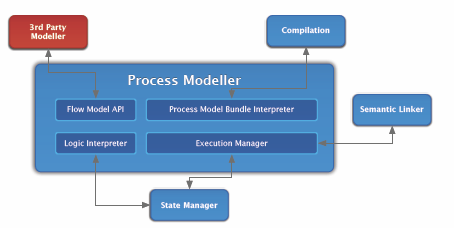
In PAA we see that provisioning model is the mapping between what Neptune had selected of parties/partners to join the services composite to execute the business process. Provisioning can serve in the knowledge tier to observe an underlay of all services been found by the semantic linker to contribute in achieving e-commerce transaction of Buying Pets. The logic of orchestration during composition of services is regarded through the implementation of intended behavior or logic model inside intention model. Provisioning model can help in building semantics of goal agent by anticipating Beliefs, Desires/Goals and Intentions. By following common sense logic which states that a predefined “Planned” steps as predicate for next action an autonomous software agent will take can be determined by abstracting logic behavior as predefined steps for optimistic approach to get final goal achieved. During agent runtime, the beliefs are updated progressively when agent senses using “common sense” technique of how much far it’s far from successful anticipation of goal function.



**PAA metadata (annotation) for neptuescript**

For our business case scenario of PetShop, the Intention “BuyPets” if been examined as an Intention Aspect, its shown later that it’s been changed in metaphor into several semantics like CreateOrder (CheckoutOrder), ValidateAccount, ShipOrder, and BillAccount these tasks or activities are been performed by external parties in our scenario, and also embrace several sub-tasks/sub-activities DeliverItems, ConfirmDelivery (receipt) and IssueInvoice. Each of these tasks and sub-tasks have ready available data and require additional data to perform its successful achievement of tasks, the required data needed are seductively defined in NeptuneScript as Needed data and Neptune is responsible to fulfill these required data. The fulfillment of requirements imposes Neptune to perform a set of actions to be taken as actuated tasks. The refactoring component is then subjected to re-write the needed “actuation” part in neptunescript thus making neptunescript more descriptive and annotated with necessary code needed to be executed. Semantic Linker is responsible to find necessary NBLO (metaobjects) needed as a translation of parsed neptuneScript at compile time.

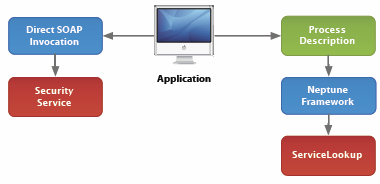




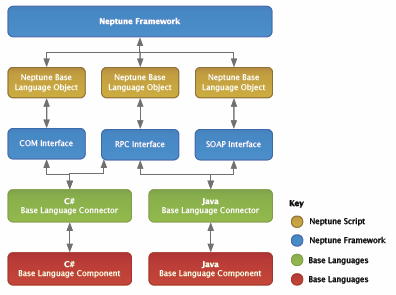


**Neptune Process Model**

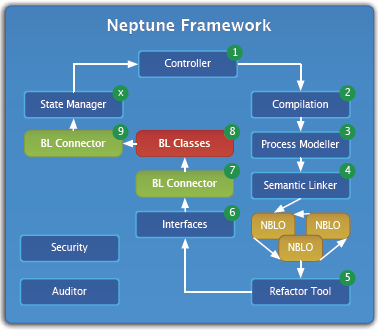
Slef-management model helps to prove autonomic system behavior for exposed compilation process functionalities. The first part is the process model; using traditional abstract process modeling, like any process oriented modeling the process model follows activity workflow principles to describe logic behavior, rules and applied through constraints to control flow. The process modeling tool used is a visual workflow modeling tool, where several shapes representing process components are orchestrated with flow controller. The tool allows feature to help adding functionality of abstracted activities to be composited later, the tool has script editor in which methods, parameters can be added through simplified expressive syntax like SmallTalk. The scripting support will help in reflection into metaobject components; NeptuneScript. After design mode, the process model can be analyzed by parsing it and checking scripts. The model will then be set into compilation process. During compilation the model is interpreted into two parts, the first part is metadata script components called NeptuneScripts and the other part is the implementation of logic control, the logic control will then marshal calls subsequently to PAA (provisioning, assurance, accounting) model which validates and controls the execution of process composite to ensure successful mapping between intentions executed and resulted system. Back to NeptuneScript components which carry descriptive context of system functionalities. Additional support should be provided to NeptuneScript to describe “requirements” needed from different application architecture layers of presentation, logic and data to support system functionalities. NeptuneScript then will be compiled by Neptune framework. During compilation, Semantic Linker component will strive to find by mimicking axiomatic semantic of the system functionality components with what’s found from searching for semantic web services in the cloud, as for the requirements part of the metadata, Semantic Linker will link to the application layers needed to be interacted as a horizontal approach. This marshaling is done through Neptune Base language objects, which are atomic components used as a means of linking and controlling data flow between partners and application layers.



NBLO’s has different interfacing methods through different protocols which helps to build a wide communication channel between application architecture layers by controlling data routes to even external domains through marshaling calls with RPC, SOAP and traditional COM.

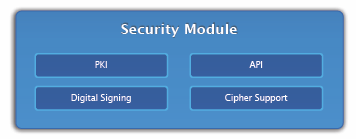


During this calls marshalling process, Neptune framework will continue reflection by logging transactions and their required data to be provided by the user, necessary portable UI components can be created at runtime to request data from different layers. By sufficing data to methods of web services, a successful composition of the application will be undertaken. A refactoring tool will help to annotate neptunescript with necessary expressions as been reported back from logging component, this will help to ensure fully descriptive portable code in neptuneScript as a metadata scripting object. The notion of self-management is well defined here by the refactoring tool through automatic update of NeptuneScript. Here the architecture assures level of autonomy because the scripting components will continue compilation until Neptune stops updating/refactoring neptunescript due to needed data

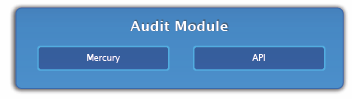


.

At runtime, Neptune framework strives to assure trust level of communication between external interaction and internal NBLO’s. This denotes to the need to control the data flow during communication and messaging. To secure this channel Neptune is rendered with new components of security and audit, security component contains several security techniques implementations to ensure privacy of data due to exposed AST (abstract syntax tree) of metadata.

****

The audit component helps to ensure tractability of the interactions made in Neptune, in particular via the semantic linking module, such that distribution of a process can take place with **knowledge** of which base language components are required to actuate the tasks of the process.

****

**Software Agent**

During performing autonomic behavior to support interaction of NBLO’s with external and internal system architectures, this behavior denotes to the Software Agent architecture which an autonomous software component senses and reacts upon other system components behavior, A software agent can proactively intercept with system architecture components due to the additional abilities of mapping real world business unit autonomy with interaction; “an agent is a software (or system component) that is *situated* in an environment, which it can perceive, and that is capable of *autonomous* action in this environment in order to meet its design objectives”. In Neptune this notion of an intelligent agent which follows BDI strategy is not exposed. It’s just embedded in the design objectives of Neptune. Our aim is to expose this notion to fit Goal needs as abstracted from intention model. One of the interesting and most significant aspects of agents is that they facilitate *cognitive* modeling (based on behavioral aspects fulfilling the purpose), as opposed to *role* modeling (based solely on purpose). If we are able to define that best describes an agent, and how it interacts with its environment and other agents, we will achieve a better contribution towards the abstraction design of intention modeling.

**Aspect Oriented Programming**

The idea of aspect oriented programming is found over the idea of aspect is a cross-cutting of concerns during software design or model design. Aspects are usually like units of system decomposition that are not functional tasks such as in our petShop scenario “no shipping for unsuccessful billing of ordered items”. Aspects cut across different components of software system. The use of AOP is that software would be better structured to help developers easier maintain code by abstracting levels of aspects from codebase. In our case the effort of examining intention model against new business aspects like Business Goals form intention aspects definition combined with intent software logic to be constructed. This challenge is quite harsh, thus to answer questions of how to explore goal aspects and construct goal graph (relation between Goal-SoftGoal and Task), and how will this graph model will help composing system process model (services, …etc) rather than rely of process oriented programming (tasks, logic,..etc). In our case the Intention aspect of BuyPets is then supported the abstract of “Order” and “OrderDetails”, then the status of order **pre-condition** as pre-validation, pre-billing and pre-shipping will help in defining where to define cut-point, and then after **post-condition**; post-validation, post-billing and post-shipping will help in weaving of concerns, so for Order.Shipping weaved to Order.Shipped and for tracking aspects using log, intention BuyPets is the source of Goal aspect ItemsShipped.

Aspect OrderShipping {

Pointcut Order.Checkout();

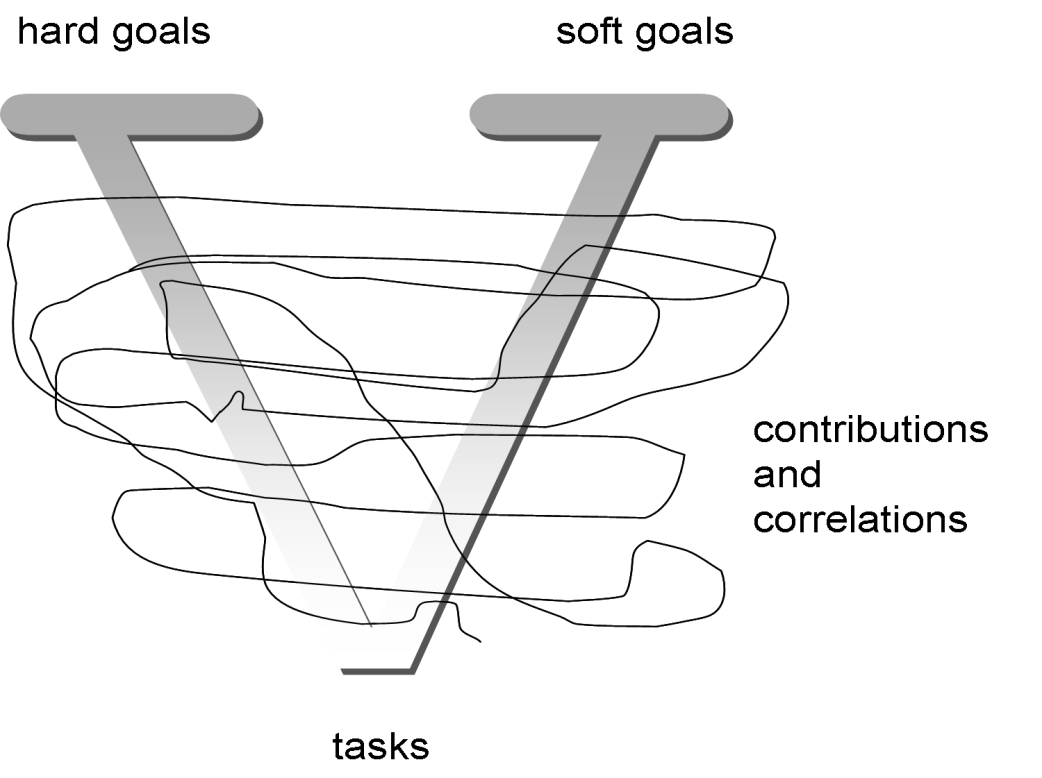
Call (Order.ValidateAccount()) ;

Call (Order.Billing());

After() returning: Order.Checkout() {

Order.Ship();

}}



**Goal Oriented Requirements**

“Tropos”is a novel agent-oriented software engineering methodology characterized by three key aspects. First, it pays attention to the activities (tasks) that precede the specification of the prescriptive requirements, like understanding **how**and **why**the intended system would meet the organizational goals. Second, it deals with all the phases of **system requirement** analysis and all the phases of **system design and implementation** in a uniform and homogeneous way, based on common mentalistic notions as those of **actors**, **goals**, **softgoals**, **plans**, **resources**, and **intentional dependencies (activities)**. Third, the methodology rests on the idea of building a model of the system-to-be that is incrementally refined and extended from a conceptual level to executable artifacts, by means of a sequence of transformational steps. One of the main advantages of the Tropos methodology is that it allows to capture not only the **what**or the **how**, but also the **why**a piece of software is developed. This, inturn, allows for a more refined analysis of the system dependencies and, in particular, for a much better and uniform treatment not only of the system functional requirements, but also of its non-functional requirements. Tropos, although not exclusively, addresses particularly well the Agent Oriented Programming, in fact, the decision of using mentalistic notions in all the analysis phases has important consequences. Tropos supports five phases of software development. The **early requirements analysis** is concerned with the understanding of a problem by studying an existing organizational setting. The output of this phase is an organizational model which includes relevant actors and their respective dependencies. Actors in the organizational setting are characterized by having goals that each single actor, in isolation, would be unable or not as well or as easily. to achieve. The goals are achievable in virtue of reciprocal means-end knowledge and dependencies. During the **late requirements analysis**, the system-to-be is described within its operational environment, along with relevant functions and qualities. This description models the system as a (relatively small) number of actors, which have a number of social dependencies with other actors in their environment. The **architectural design phase** deals with the definition of the system global architecture in terms of subsystems that are represented as actors and their data dependencies that are represented as actor dependencies. The **detailed design phase** aims at specifying each architectural component in further detail in terms of inputs, outputs, control and other relevant information. Finally, during the **implementation phase**, the actual implementation of the system is carried out, consistently with the detailed design.

Neptune NBLO objects are the software autonomous agents, which are dynamically interact through Neptune process model (neptunescript), thus by annotating neptunescript with necessary required support for Goal modeling will lead to intelligent behavior of NBLO’s. The refactoring component is then required to draw the route as planned advise for the software agent of how will it find the right artifacts to fulfill Goals.

**This research work objective is to Abstract Goal Oriented Requirements from Intention Model using semantic (axiomatic, denotational) analysis of Intention model rather than syntactic analysis, then use this notion with aid of AOP to describe Goal Oriented Requirements as a V-Graph which genders the decomposition of goals into subgoals, and softgoals, these decompositions take place if the denotaional semantics in the intention model is rich enough. (Goal of buying pets is then pets are shipped successfully, but in meantime this goal passes into several aspects like order creation, account validation, billing then finally shipping, so for aspect shipping items, is the only aspect that really suffice the intention of buy pets, while other aspects are only denotational or means to reach that goal, pre-conditions of this goal is that all aspects should be true (validate account, billing) in order to achieve last aspect.**

Define DeliverPets as goal

{

Features{

Feature **Customer** as Goal**Owner**;

Feature **‘To deliver shipped pets’** as Goal**def**;

Feature **High** as Goal**priority**;

}

}

**Goal Abstraction (metadata to annotate neptunescript)**



**Goal Oriented Requirements Abstraction**



Denotationl Semantics Analysis higher level goal as influenced by intention F(P)=D where intentions are been denoted by concept of processorder o be then achieved by its decomposition into requirements of processing order which are checkoutorder, validateorder, billaccount then shipaccount; all these constraints should be fulfilled for the order to be shipped





**Abstraction of Aspects**

Pets: Product

[BuyPets, SellPets, ExchangePets]: Intention

PetOrder: Order

A good example of using metaprogramming technique like “reflection” to monitor objects and perform reactive actions at runtime is SmallTalk, in SmallTalk everything (expression) is an object (variables,..etc) and objects are in the meta level talk to each other (communicate) through messaging so for a decleration of a var like a =b +c for ex. A,b , c, = and + are objects (even operands are objects), so when syntax is parsed it is compiled then into meta objects before last target language of interpretation. These meta objects talk to each other via messaging like + operand which is denoted by object (add as method), send message to objects b and c to take their values and add them then back to (reflection) operand a to put the result, which is the Goal of the statement. This goal is been achieved by the function + (Add) and two inputs a and b.

Identifier = (Letter | "\_"), {(LetterOrDigit | "\_")};

Reference = Identifier;

ConstantReference =   
                "nil"   
                | "false"   
                | "true";

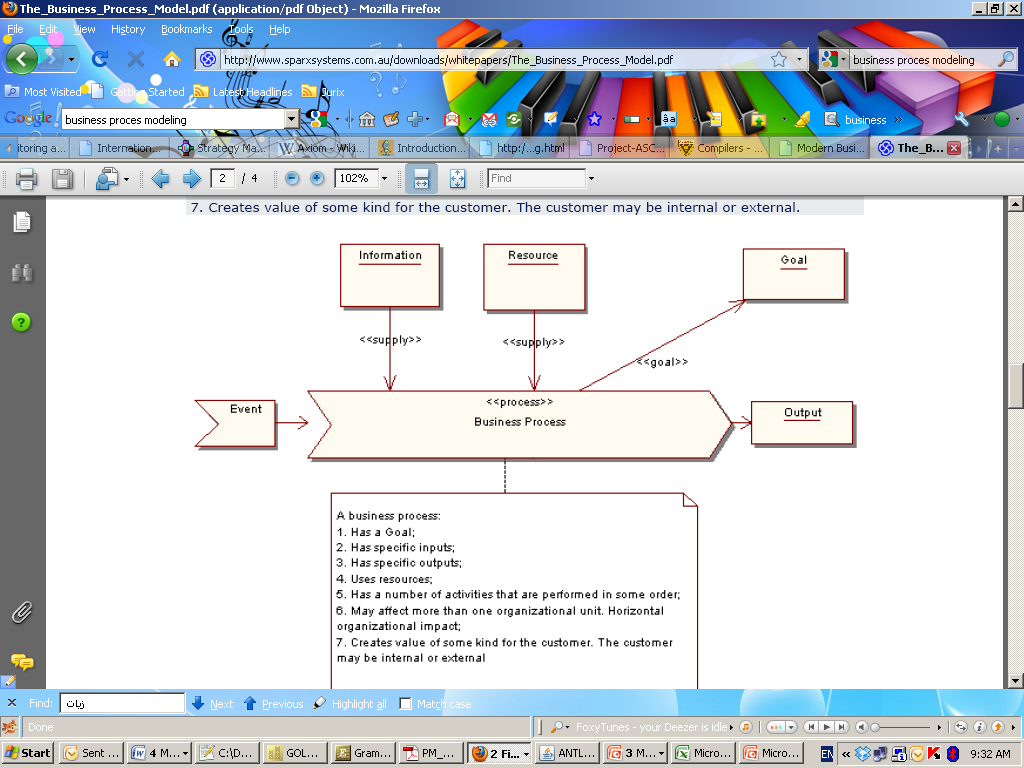
PseudoVariableReference =   
                "self"   
                | "super"   
                | "thisContext";   
        (\* "thisContext" is not defined by the ANSI Standard, but is widely used anyway \*)

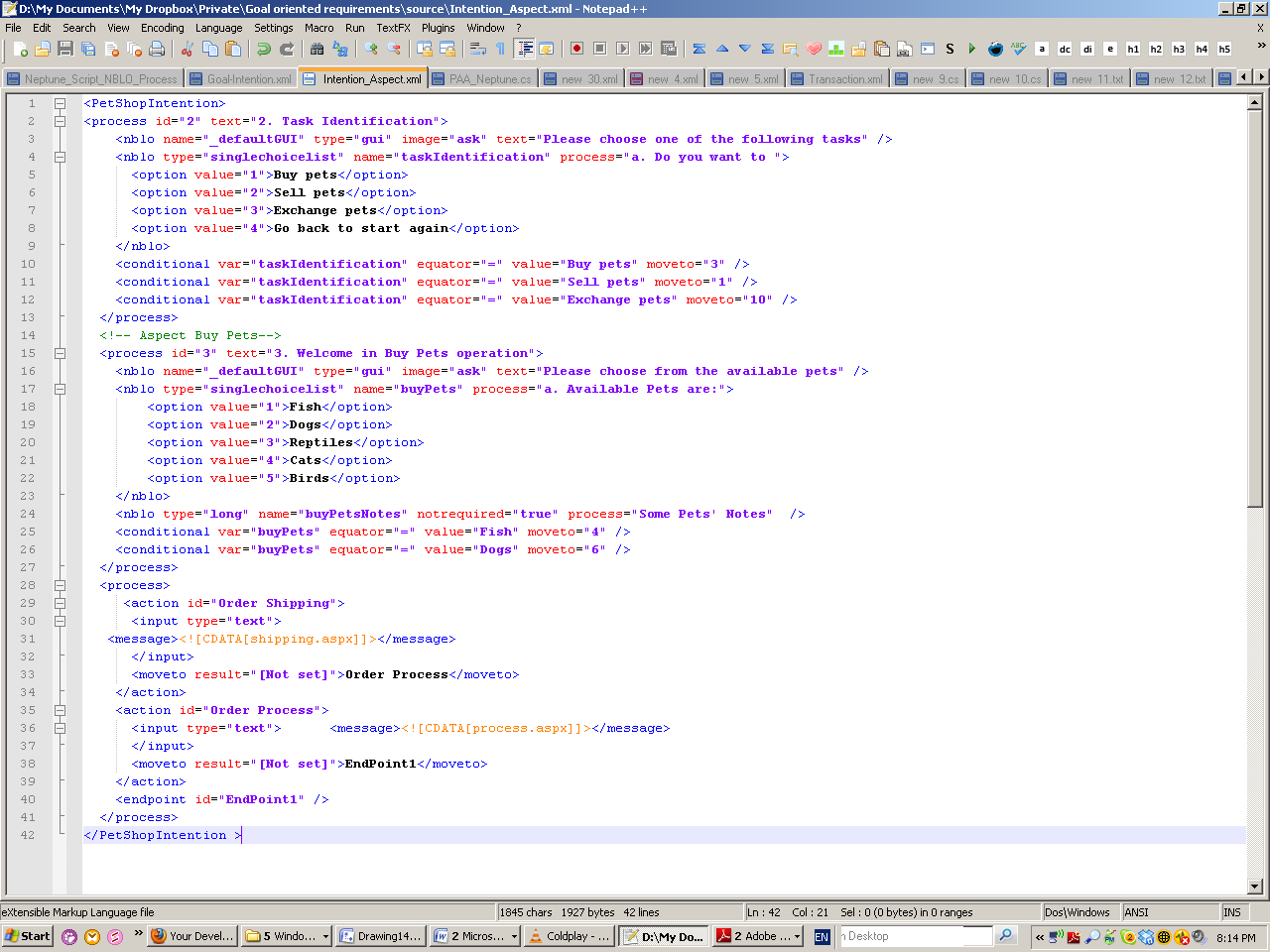
**Formal EBNF Specification of Smalltalk Syntax**

**Intention Modeling**

// details to be added for how to analyze intention model based on methodologies of semantic and syntactic analysis

**Business Process Modeling**



**Intention Model**



Intention Wiki Editor (IDE)

process process Order

{

tasks

{

if ( validateAccount == false )

end;

orderBilling giving orderID ;

orderShipping ( orderID ) ;

orderProcess ( orderID ) ;

}

}

process processOrder

{

tasks

{

validateAccount ;

if ( ! State . orderID . valid )

end ;

orderBilling ;

orderShipping ( State . orderID ) ;

//adapted process

checkDelivery ;

if ( State . orderID . isUKBased )

orderProcess ( State . orderID ) ;

}

}

task OrderBilling with NString ordered

{

requirements

{

//need data

needAddress : require orderID . BillingAddress ;

needPostcode : require orderID . BillingPostcode ;

needBand : require orderID . BillingBand ;

//need to store this data

saved : require Database . Billing ( orderID ) ;

}

//as processed by the semantic l inker

actions

{

needAddress , needPostcode , needBand via nbloOrderBilling ;

saved via nbloSaveOrderBilling ;

}

}

task OrderShipping with NString orderID

{

requirements

{

//need data

needAddress : require orderID . ShippingAddress ;

needPostcode : require orderID . ShippingPostcode ;

needBand : require orderID . ShippingBand ;

//need to store this data

saved : require Database . Shipping ( orderID ) ;

}

//as processed by the semantic l inker

actions

{

needAddress , needPostcode , needBand via nbloOrderShipping ;

saved via nbloSaveOrderShipping ;

}

}

**Neptune Process Model Metadata (NeptuneScript)**

**Neptune Process Model (NeptuneScript) Syntax Requirements**

NeptuneScript model (process model) requires a clear defined description of abstract cross cut concerns for Concepts and their features, then a definition of Aspects



****

