Journal of Universal Computer Science, vol. 13, no. 13 (2007), 2035-2075
submitted: 30/9/06, accepted: 15/11/07, appeared: 28/12/07 © J.UCS

From Theoretical e-Barter Models to Two Alternative Implementations Based on Web Services

Mario Bravetti, Adalberto Casalboni
(Università di Bologna, Dipartimento di Scienze dell'Informazione
Mura Anteo Zamboni 7, 40127 Bologna, Italy
e-mail: `bravetti@cs.unibo.it`)

Manuel Núñez, Ismael Rodríguez
(Dpt. Sistemas Informáticos y Computación, Universidad Complutense Madrid
C/ Profesor José García Santesmases s/n, 28040 Madrid, Spain
e-mail: `{mn,isrodrig}@sip.ucm.es`)

Abstract: An e-barter system is an e-commerce environment where transactions do not necessarily involve money. They are multi-agent systems where agents perform exchanges of resources on behalf of their respective users. Besides, their structure is based on a tree of markets. In this paper we show how to develop suitable *designs* for this kind of systems by means of *web services* by using WS-BPEL. Since the formal specification abstracts most practical details, the development of such design definition requires to face several challenges. We present two alternative designs that both comply with the formal specification.

Key Words: formal methods, e-barter, web services.
Category: D.2.1, F.3.1, K.4.4

1 Introduction

Among those areas where the development of Computer Science has changed our society during the last years, the relevance of e-commerce technologies is remarkable. New mechanisms to perform transactions have appeared and they entail new challenges. Since e-commerce systems dramatically affect user's possessions, their reliability is specially important for their success. Similarly to other domains, formal specifications allow to predict relevant properties of e-commerce systems and they provide a reference of ideal behavior for developing an implementation. In order to provide handleable and unambiguous models, formal languages abstract from some low level details that are considered irrelevant for the system description. In spite of the gap between specification models and final implementations, specifications provide a developer with a model of ideal behavior of the system that may be useful not only in the *analysis* phase of a project but also during the whole development process.

Among existing applications of formal methods to the area of e-commerce (e.g. [Rao, 1996, Hindriks et al., 1998, Probert et al., 2003, Cavalli and Maag, 2004, Núñez et al., 2005b]), here we consider the formal definition of e-barter systems formerly introduced in [López et al., 2002] and extended in [López et al., 2003, Núñez et al., 2005a]. This kind of systems are characterized because transactions do not necessarily involve money. Actually, they provide a generalization of the classical e-commerce model because they can represent systems where exchanges may or may not involve the use of money. Users connected to the system exchange goods according to their respective preferences. Exchanges are performed until the system reaches a kind of optimal distribution of resources with respect to the user requests. In order to avoid critical bottlenecks in the system performance, a single global market is replaced by a hierarchical tree-like structure of markets, where parallelism can be exploited allowing local markets to progress independently to each other. Following this structure, users are connected to different local markets in the leaves of the tree according to proximity reasons. Exchanges are performed inside each local market until it reaches an optimal distribution. Then, the market *becomes* an agent that acts as representative of the users that traded inside it. This agent exchanges resources on behalf of its users with other (representative) agents in a *higher level* market until this higher order market reaches an optimum. The mechanism is repeated until the global market, that is the one at the top of the hierarchy, reaches an optimum. The behavior of this system is formally defined in terms of a process algebraic notation. This notation allows us to speak about a specification that is *parametrical* on the market structure: Depending on the term constructed to define the actual system, a different stratification of markets into levels is created. The system definition allows to formally reason about the behavior of the system and predict some relevant theoretical results. In particular, as it is shown in [Núñez et al., 2005a], if an adequate amount of information is exchanged among the different levels, the final distribution of resources is *optimal* from the point of view of the users. That is, the hierarchical system reaches the same kind of distribution that can be reached by a non-hierarchical system where a single central market would embrace all the agents (i.e. the *economic* efficiency of a hierarchical market matches that of a non-hierarchical market).

In this paper we face the problem of developing a design of the e-barter system which may constitute the basis for an efficient implementation.[1] The idea of our e-barter system is to be a world-wide system structured in geographical sublevels. Communicating geographically spread participants in any system (in this case, an e-barter system) requires a communication network, so Internet is the natural choice. Distributing markets world-wide in servers as services (inside the related geographic area) is motivated by the improvement in the communication efficiency (closeness to service clients) as well as the benefits of distributing the

[1] This paper is an extended and revised version of the FSEN'05 paper [Bravetti et al., 2006].

computation on servers: Markets are executed on different servers thus exploiting the inherent parallelism in the system structure.

Web services related technologies are a set of middle-ware technologies for supporting *Service Oriented Computing* [Huhns and Singh, 2005]. One of the main goals of this paradigm consists in enabling the construction of a network of integrated collaborative applications regardless of the platform and the development language. Moreover, web Services and the related notion of *orchestration* of services constitute a suitable *conceptualization semi-formal model* allowing to move from an abstract model to an implementable distributed model. Orchestration of web services is supported e.g. by WS-BPEL [Andrews and Curbera, 2004] (*Business Processes for Web Services*), which is a language for describing web-service behavior (workflow) in terms of invokations to other web-services. Orchestration technologies like WS-BPEL can be used for the design of systems, by defining web-services in terms of composition of other web-services. They, at the same time, constitute an adequate tool for the design of service oriented systems (and the analysis of the relationship with the specification) and make it possible to produce models directly usable in the implementation (since they are executable by means of engines or compilable). On the other hand, they allow to define communicating partners in the Internet in a modular, structured, and semi-formal way. In particular, a web service is identified by a URI, and public interfaces and bindings are defined by using XML so that their definition can be discovered by other software systems [W3C, 2006]. Web services allow to publish services on the internet by means of a naming service called UDDI [UDDI, 2006] (*Universal Description, Discovery and Integration*). XML based WSDL technology [Christenses et al., 2001] (*Web Services Description Language*) is used to describe services inside UDDI naming servers and making them accessible at a certain location by means of an XML based protocol called SOAP [SOAP, 2006] (*Simple Object Access Protocol*).

Concerning exploitation of modular capabilities of web-services, in the paper we show that, by using local UDDI servers, thus defining an independent *namespace* for each market environment (that includes agents trading in it and also clients for markets at the lowest level), we can give a single *uniform* design for all clients (and similarly for all markets and all agents): Once deployed on servers and "executed", these designs will correctly interact with each other according to a tree structure because the same names in different local UDDI services are linked to different physical addresses (the system structure, e.g. the connection among markets, is defined by adequately binding names in each local UDDI service).

The representation of the e-barter system as orchestration of web services requires to address several practical challenges and forces the developer to define more in the detail the system architecture and behavior: The entities involved (which as we will see can be both concrete, i.e. real services on the internet, or abstract, i.e. just representing activities made by human beings), the flow of interaction among these entities, the kinds of data exchanged, the time for message exchange, timeout for receiving requests, etc. Starting from the formal system specification we will develop two alternative designs of a possible implementation, where entities and flow interactions are defined in such a way as to obtain an efficient implementation which exploits system parallelism of different local markets expressed as distributed services (instead of using a single centralized market). The first proposed design permits to reach distributions of resources that are optimal for the clients of the system (more precisely, *Pareto optimal* distributions) provided that all agents willing to participate in a market connect to the market before a given *timeout* is reached. This is achieved by making the design to fulfill some requirements that are known to be sufficient to provide optimal distributions. As we commented before, these requirements were identified as a property of the *specification* in [Núñez et al., 2005a]. Hence, this is an example of how we can take advantage of a formal model of the specification and its known properties to construct better designs. The second design allows to improve the system performance at the price of not fulfilling these requirements, that is, at the price of losing optimality of the solution in certain cases.

The relation between the specification, defined by means of a process algebra, and each design, defined via web services, is twofold. On the one hand, the specification provides us with an abstract model that allows to construct the design in an unambiguous way. Moreover, non-trivial formal properties, that are easier to find in a simple model such as the specification, allow the designer to know the consequences of different design choices (in our case, optimality of distributions *vs* performance). On the other hand, practical issues found during the construction of the design lead to reconsider some aspects of the specification itself. Since the specification model presented in [López et al., 2002, López et al., 2003, Núñez et al., 2005a] implicitly assumed the requirements leading to optimal distributions, it was not a suitable specification of our second alternative design, which does not fulfill them. Moreover, it was not for the first one either, because all agents were assumed to get connected to higher markets and this does not conform to this design. Hence, a new specification, adapted to the feedback provided during the design phase, was required. In this paper, a generalized specification that is suitable for both alternative designs is proposed. In particular, requirements leading to optimality are removed, thus leaving freedom to the design to either fulfill them or not. In technical terms, the new operational rules of the specification produce more *non-determinism* at some points. This leaves the task of fixing a criterion to choose among the several available choices to the design. In this way, several designs (in particular, those presented in this paper) may *particularize* the specification in different directions. In addition, this specification is *extended* to include additional aspects that actually are needed in both target designs. In particular, the fact that agents do not participate in a market is recorded, in such a way that they may participate in the next execution of the system and the structure of markets is kept, is explicitly represented in the new specification. Hence, in this paper the specification is defined in higher detail in such a way that it is closer to the behavior considered in the designs; it actually particularizes the former specification.

This is an example of how specifications and designs can successively influence each other during the development process of a system in such a way that, after each iteration, new designs that comply with the new specifications are constructed. This process of mutual discovery implicitly assumes that specifications and designs are flexible entities that may evolve as long as the developer understanding of the target system improves. Thus, instead of an automatic process, in this paper we must speak about a manual redefinition process.

The rest of the paper is structured as follows. In Section 2 we sketch the formal model defining e-barter systems. In Section 3 we present the main concepts on which the design with web services is based and we discuss how it is obtained from the specification. In Section 4 we present the web-service design of the e-barter system that yields global optimality if some requirements hold. In Section 5 we present the alternative more efficient design. Finally, in Section 6 we present our conclusions.

2 Formal specification of multi-level e-barter systems

In this section we briefly describe e-barter systems and introduce their formal specification. An e-barter system is a multi-agent system where agents exchange resources on behalf of their respective users. Since agents must perform exchanges according to the preferences of users, we need a suitable notation to denote preferences. We use *utility functions*. The input of a utility function is a *basket* of resources and the output is a *numerical value* denoting the preference on this basket. Let f_A be the utility function of an agent A. Let $\bar{x}$ be a basket with 2 apples and 1 euro, and $\bar{y}$ be another basket with 1 apple and 2 euros. If $f_A(\bar{x}) > f_A(\bar{y})$ then A *prefers* $\bar{x}$ to $\bar{y}$. A possible utility function showing that behavior is $f_A(apples, euros) = 2 \cdot apples + 1 \cdot euros$. Let us suppose there is another agent B whose utility function is $f_B(apples, euros) = 2 \cdot apples + 3 \cdot euros$. Then, if the agents A and B perform an exchange where agent A gives 1 euro to B and B gives 1 apple to A, then both utility functions return higher values after the exchange, that is, both agents *improve*.

An e-barter system performs *fair* exchanges, that is, exchanges where at least one agent improves and none of them worsens. When no more fair exchanges are available, the market is *completed*, that is, it reaches a configuration that

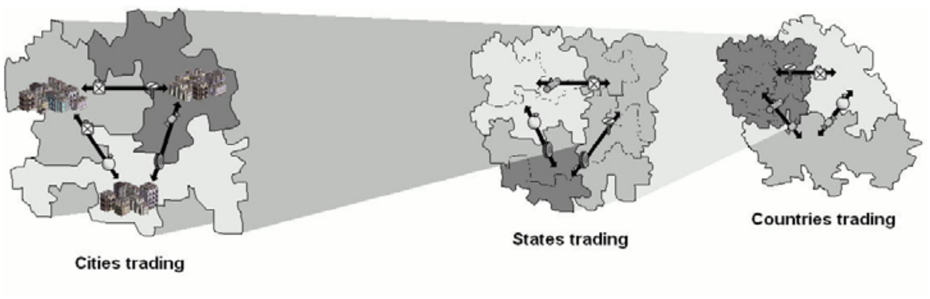

Figure 1: The hierarchical structure of markets.

cannot be improved. Instead of using a single market where all agents in the system exchange resources, agents will be grouped in local markets according to proximity reasons. For example, agents belonging to the same city are put together until their markets are completed. Then, they use some *representatives* to exchange resources in a higher order market involving several cities. After these are completed, new representatives exchange resources in a higher market, and so on. The general structure of the system is depicted in Figure 1. This scheme has several advantages. On the one hand, it promotes that exchanges are performed as near as possible, which reduces shipping costs. On the other hand, the hierarchical structure improves the computational efficiency of the system by exploiting the parallelism of the structure (several markets, instead of a single market, work in parallel) and increasing the system robustness (if a market is temporarily out of service, the rest of the structure is still operative).

Initially, customers willing to participate in an e-barter system are *represented* by (electronic) agents. These agents are provided with two parameters: The *basket of resources* that the customer is willing to exchange and a *utility function*. Such agents trade in the most local market whose area includes the location of the the related customers.

Then the e-barter system works according to the following algorithm:

(1) Agents exchange goods inside their local market. A multilateral exchange will be made if (at least) one of the involved agents improves its utility and none of them decreases its utility. This is repeated until no more exchanges are possible. In this case we say that the local market is *completed* (or *saturated*).

(2) Once a market is completed, their agents are combined to create a new agent which will trade in the higher level market whose area include the location of the current market (this happens unless we are in the top-level market). This agent behaves as a representative of the combined agents. The new agent will

have as basket of resources the union of the baskets corresponding to each agent. Its utility function will encode the utilities of the combined agents. This utility function imposes that resources obtained by the representative are such that they can delivered among its users in such a way that no user worsens with respect to the previous distribution. If this condition holds, it returns the addition of utilities of each represented user. The construction of this utility function will be formally defined in the forthcoming Section 2.2. The *higher order* agent participates in a higher level market.

(3) The exchanges performed in the higher level market are performed starting again from (1), unless we are in the top-level market. In this case the process is finished and the distribution of goods obtained is the final one.

Note that during the whole process, after the completion of any market, the formal model keeps track of distribution of goods to the several customers by propagating such information in a top-down way through the tree of markets until it arrives to the leaves of the tree (i.e. to the agents directly representing customers).

As we will see in Section 3, the design of the e-barter system will address several practical issues that are beyond the detail level considered in the previous description. Some of these issues will make us to *reconsider* the previous scheme. For example, requiring that all subagents are connected to a market before it becomes completed is actually needed to provide final optimal distributions, but this requirement might not be feasible in practice. For instance, if an agent is temporally out, then it may block the rest of agents. So, when a market becomes a new higher order agent (see step (2)), in some situations it will be able to become a representative of only *some* agents. These changes will affect the operational semantics of the formal language (presented in Section 2.2). In order to endow the design(s) with enough freedom to choose which agents are represented by new agents, according to their own criteria, operational rules of the specification will be non-deterministic, i.e., the specification will allow *any* choice at his point. Note that these changes show that the relation between the formal specification (i.e., the model of the analysis of the system) and the web services definition we construct from it (i.e., the design and implementation) is two-fold.

2.1 A brief introduction to the formal model

Next we briefly introduce the formal representation of e-barter systems. The formal definition of an e-barter system is made by means of a specification language that was explicitly developed to define this kind of systems. An e-barter system is formally given by a syntactical term in the syntax of this language. Moreover, this term explicitly defines the structure of the tree of markets, i.e., how agents and markets are connected to higher order markets. The semantics of the language implicitly define the behavior of an e-barter system in a formal fashion. The operational semantics can be found in the next section. Even though this language uses a process algebraic notation (mainly when defining the operational rules) it does not need the usual operators appearing in this kind of languages (choice, restriction, etc). In fact, our constructions remind a parallel operator as the one presented, for example, in the process algebra CCS [Milner, 1989].

Definition 1. A *market system* is given by the following EBNF:

$$\begin{array}{lcl} MS & ::= & ms(M) \\ M & ::= & A \mid \mathtt{uncomp}(M, \ldots, M) \\ A & ::= & (id, S, T, u, \overline{x}) \\ S & ::= & [\,] \mid [A, \ldots, A] \\ T & ::= & [\,] \mid [M, \ldots, M] \end{array}$$

where for all terms of the form A, u is a utility function, $\overline{x}$ is a basket of resources, and $id \in \{id_1, id_2, id_3, \ldots\}$ is an *identification* symbol. We assume that the identification symbol of each term of the form A is unique. □

First, in order to avoid ambiguity of the grammar, we annotate market systems with the terminal symbol ms. Intuitively, the market $M = (id, S, T, u, \overline{x})$ (that is, $M = A$) represents a *completed* market, that is, a market where no more exchanges can be performed among its agents. Let us note that in this case the market represents an *agent* that will be able to make transactions with other agents in a higher market. In the previous expression, id denotes the unique identification symbol of the agent denoted by M, u denotes the utility function of M, and $\overline{x}$ represents the basket of resources owned by M. We consider that there are p different commodities,[2] that is $\overline{x} \in \mathbb{R}_+^p$, and that the amount of *money* is placed in the last component of the tuple. We will assume that `UtilFuncs` denotes the set of all utility functions, that is, functions with the form $\mathbb{R}_+^p \longrightarrow \mathbb{R}_+$.

Regarding the first two arguments of M, that is S and T, they denote those markets/agents that are below the agent in the hierarchical tree. S represents those agents that are actually represented by the agent (that is, it can exchange their resources in order to reach better baskets for them), while T represents those lower markets/agents that are not. As we commented before, sometimes agents will represent only a subset of the agents that are below them for efficiency reasons; the lists S and T allow to clearly separate them (only S was used

[2] We are assuming that all the items are *goods*. Nevertheless, agents could also trade *bads*. For example, a customer would be willing to give an apple pie if he *receives* minus s brown leaves in his garden. However, bads are usually not considered in microeconomic theory, as they can be easily turned into goods: Instead of considering the amount of leaves, one may consider the absence of them.

in definition of the language given in [Núñez et al., 2005a]). Depending on the values of S and T we consider two possibilities. Either both S and T are empty lists or either of them is not. In the first case we have that M represents an *original* agent, that is, a direct representative of a customer (note that a single agent is trivially completed since there is nobody to deal with). In the second case, that is, if either $S = [A_1, \ldots, A_n]$ with $n \geq 1$ or $T = [A'_1, \ldots, A'_m]$ with $m \geq 1$, then we have that M represents an agent associated with the (possible higher order) agents $A_1, \ldots, A_n$ belonging to a completed market. Let us note that if $S = [\,]$ and $T \neq [\,]$ then the agent will idle.

The second possible syntactic form of M, $\texttt{uncomp}(M_1, \ldots, M_n)$, represents an *uncompleted* market consisting of the markets $M_1, \ldots, M_n$. Let us remark that in this case some of the sub-markets may be completed.

Next we present an example showing how an e-barter system may be constructed. In this example we will also (informally) introduce the operational transitions of the language.

Example 1. Let us consider a system including six agents $A_i = (id_i, [\,], [\,], u_i, \overline{x_i})$, for $1 \leq i \leq 6$. We suppose that these agents are grouped into three different markets. Initially, these markets are uncompleted, so we make the following definitions:

$$M_1 = \texttt{uncomp}(A_1, A_2)$$
$$M_2 = \texttt{uncomp}(A_3, A_4)$$
$$M_3 = \texttt{uncomp}(A_5, A_6)$$

Let us consider that the first two markets are linked, and the resulting market is also linked with the remaining market M_3. We should add the following definitions:

$$M_4 = \texttt{uncomp}(M_1, M_2)$$
$$M_5 = \texttt{uncomp}(M_4, M_3)$$

Finally, the global market is defined as $M = ms(M_5)$. This hierarchical structure is graphically presented in Figure 2, top-left (uncompleted markets are represented by a single square).

Following the philosophy explained in the previous section, transactions will be made within a market only among completed sub-markets. So, initially only M_1, M_2, and M_3 are allowed to perform transactions (as we remarked before, original agents are trivially completed).

We will use the symbol $\rightsquigarrow$ to denote exchange of resources. Let us suppose that, after some exchanges, M_1 gets completed. That is, there exists a sequence of exchanges $M_1 \rightsquigarrow M_1^1 \rightsquigarrow M_1^2 \cdots \rightsquigarrow M_1^n = M'_1$ such that $M'_1 \not\rightsquigarrow$. In this case, the market grouping the first two agents should be labeled as completed. So, the agents effectively perform all the achieved transactions becoming A'_1 and A'_2, respectively. Then, the first market becomes $(id', [A'_1, A'_2], [\,], f(u_1, u_2), \overline{x_1} + \overline{x_2})$, where id' is a fresh identification symbol and f is a function combining utility

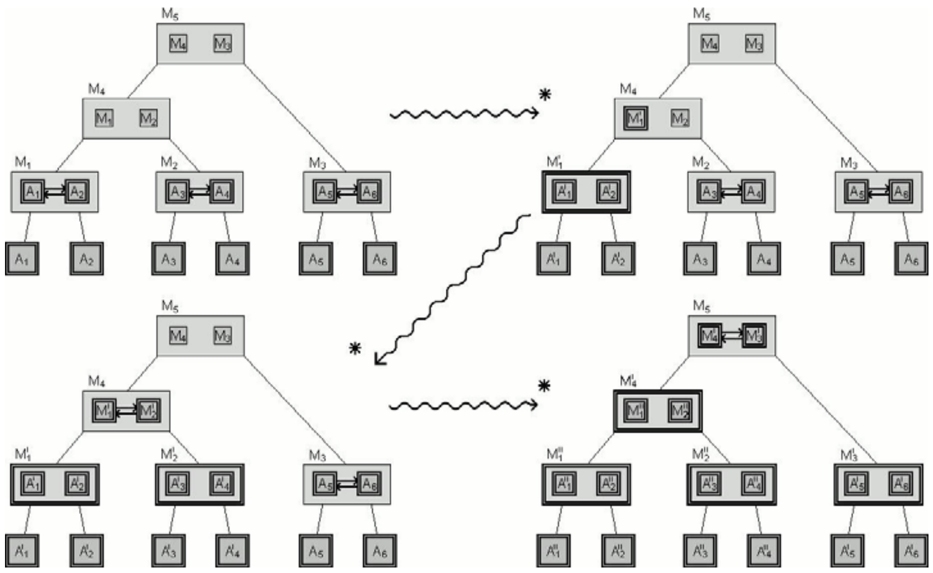

Figure 2: A market system and some operational transitions.

functions which works as described in the algorithm presented in the previous section (step 2); the construction of this function is formally defined in the next section. In parallel, M_2 will have a similar behavior.

Once both M_1 and M_2 get completed, transactions between them will be allowed. Note that these transactions (inside the market M_4) will be performed according to the new utility functions, $f(u_1, u_2)$ and $f(u_3, u_4)$, and to the new baskets of resources, $\overline{x_1} + \overline{x_2}$ and $\overline{x_3} + \overline{x_4}$.

The process will iterate until M_5 gets completed. At this point, the whole process finishes. □

Note that if we assume that markets can become completed even if not all the submarkets are completed yet and that new higher order agents can represent only a subset of agents, then other sequences of interaction may occur in the previous example. Despite the presentation of the formal model in [López et al., 2002, López et al., 2003] does not consider these features, the formal model considered in this paper includes them. As we said before, the goal of adding them to the system is to take into account some efficiency issues that will be addressed in the design phase.

2.2 Operational semantics of the specification model

Next we present a brief description of the operational semantics of the specification language used to specify e-barter systems. A full description can be found in [Núñez et al., 2005a], though a few details have been changed to properly represent some practical requirements that were discovered during the construction of the system design.

In order to simplify forthcoming operational rules we introduce the following notation to deal with utility functions. Utility functions associated with original agents (that is, $A = (id, [\ ], [\ ], u, \overline{x})$) will behave as explained before: Given a basket of resources, a single value denoting the utility of the basket is returned. That is, $u(\overline{z})$ indicates the relative preference shown by A towards the basket of resources $\overline{z}$. Nevertheless, if $A = (id, [A_1, \dots, A_n], T, u, \overline{x})$ then we will consider that, in addition to its usual meaning, the utility function also keeps track of how a basket of resources is distributed among the (possible higher order) agents $A_1, \dots, A_n$. That is, $u(\overline{z}) = (r, \overline{z_1}, \dots, \overline{z_n})$, where r still represents the utility, while $\sum \overline{z_i} = \overline{z}$ and $\overline{z_i}$ denotes the portion of the basket $\overline{z}$ assigned to A_i. Overloading the notation, if we simply write $u(\overline{z})$ we are referring to the first component of the tuple, while $u(\overline{z}).i$ denotes the $(i+1)$-*th* component of the tuple. Later we will see how the utility functions following this pattern are actually constructed.

In the next definition we present the *anchor case* of our operational semantics. In order to perform complex exchanges, agents should first indicate the barters they are willing to accept.

Definition 2. Let $A = (id, S, T, u, \overline{x})$ be a completed market. The *exchanges* the agent A would perform are given by the following operational rules:

$$\frac{u(\overline{x}+\overline{y}) \geq u(\overline{x}) \ \wedge \ (\overline{x}+\overline{y}) \geq \overline{0}}{(id,S,T,u,\overline{x}) \xrightarrow{\overline{y}} (id,S,T,u,\overline{x}+\overline{y})}$$

$$\frac{u(\overline{x}+\overline{y}) > u(\overline{x}) \ \wedge \ (\overline{x}+\overline{y}) \geq \overline{0}}{(id,S,T,u,\overline{x}) \xmapsto{\overline{y}} (id,S,T,u,\overline{x}+\overline{y})}$$

where $\overline{y} \in \mathbb{R}^p$, being p the number of different commodities. □

Let us remark that $\overline{y}$ may have negative components. Actually, these tuples will contain the barters offered by the agent. For example, if $\overline{y} = (1, -1, 0, -3)$ fulfills the premise then the agent would accept a barter where it is offered one unit of the first product in exchange of one unit of the second good and three units of money. Regarding the rules, the first premise simply indicates that the agent would not decrease (resp. would increase) its utility. The second premise indicates that the agent does not run into *red numbers*, that is, an agent cannot offer a quantity of an item if it does not own it. Thus, a transition as $\longrightarrow$ denotes that the agent does not worsen; a transition $\longmapsto$ denotes that the agent does improve. Next we show how offers are combined.

$$\frac{\exists\, k \in I : M_k \xmapsto{\overline{y_k}} M'_k \;\wedge\; \forall\, i \in I, M_i \xrightarrow{\overline{y_i}} M'_i \;\wedge\; \mathtt{valid}(M, \mathcal{E})}{M \overset{\mathcal{E}}{\leadsto} \mathtt{uncomp}(M'_1, \ldots, M'_n)}$$

where • $M = \mathtt{uncomp}(M_1, \ldots, M_n)$ and $M_i = (id_i, S_i, T_i, u_i, \overline{x_i})$ for all $1 \leq i \leq n$
• $\mathcal{E} \in (\mathbb{R}^p_+)^{n \times n}$ and $I = \{s_1, \ldots, s_r\} \subseteq \{1, \ldots, n\}$
• $M'_i = \begin{cases} M_i & i \notin I \\ (id_i, S_i, u_i, \overline{x_i} + \overline{y_i}) & \text{otherwise} \end{cases}$
• $\overline{y_i} = \sum_j \mathcal{E}_{ji} - \sum_j \mathcal{E}_{ij}$, for any $i \in I$

Figure 3: Operational rule for the exchange of resources in uncompleted markets.

Definition 3. Let $M = \mathtt{uncomp}(M_1, \ldots, M_n)$ be an uncompleted market and $I = \{s_1, \ldots, s_r\} \subseteq \{1, \ldots, n\}$ be a set of indexes denoting the completed markets belonging to M (that is, for any $i \in I$ we have that $M_i = (id_i, S_i, T_i, u_i, \overline{x_i})$). We say that the matrix $\mathcal{E} \in (\mathbb{R}^p_+)^{n \times n}$ is a *valid exchange matrix for* M, denoted by $\mathtt{valid}(M, \mathcal{E})$, if the following conditions hold:

– For any $1 \leq i \leq n$ we have $\sum_j \mathcal{E}_{ij} \leq \overline{x_i}$,

– for any $1 \leq i \leq n$ we have $\mathcal{E}_{ii} = \overline{0}$, and

– for any $1 \leq i, k \leq n$ such that $k \notin I$ we have $\mathcal{E}_{ki} = \overline{0}$ and $\mathcal{E}_{ik} = \overline{0}$.

□

First, let us note that the notion of *valid* matrix is considered only in the context of uncompleted markets: If a market is already completed then no more exchanges can be performed. Second, only completed markets belonging to an uncompleted one may perform exchanges among them. This restriction allows to give priority to transactions performed by *closer* agents belonging to uncompleted sub-markets. Regarding the definition of *valid* matrix, the components of matrixes $\mathcal{E}$ are baskets of resources (that is, elements belonging to $\mathbb{R}^p_+$). Thus, $\mathcal{E}_{ij}$ represents the basket of resources that the market M_i would give to M_j. The condition $\sum_j \mathcal{E}_{ij} \leq \overline{x_i}$ indicates that the total amount of resources given by the market M_i must be less than or equal to the basket of resources owned by that market. Let us also comment that an exchange does not need to include all of the completed markets. That is, if we have an exchange where only r' markets participate, then the rows and columns corresponding to the remaining $r - r'$

completed markets will be filled with $\overline{0}$. Besides, the rows and columns corresponding to the $n-r$ uncompleted markets will be also filled with $\overline{0}$. Finally, let us note that the validity of an exchange matrix must be checked when the exchange defined by the matrix is to be performed, i.e., we consider the current state. Obviously, the validity of a given matrix remains unchanged as long as no exchange is performed.

Next we introduce the rules defining the exchange of resources. Intuitively, if we have a valid exchange matrix where at least one of the involved agents improves and no one worsens then the corresponding exchange can be performed.

Definition 4. Let $M = \texttt{uncomp}(M_1, \ldots, M_n)$ be an uncompleted market and $I = \{s_1, \ldots, s_r\} \subseteq \{1, \ldots, n\}$ be a set of indexes denoting the completed markets belonging to M (that is, for any $i \in I$ we have that $M_i = (id_i, S_i, T_i, u_i, \overline{x_i})$). The operational transitions denoting the exchange of resources that M may perform are given by the rule shown in Figure 3. We say that M is a *local optimum*, denoted by $M \not\rightsquigarrow$, if there do not exist M' and $\mathcal{E}$ such that $M \overset{\mathcal{E}}{\rightsquigarrow} M'$. □

The operational rule presented in Figure 3 is applied under the same conditions appearing in the definition of a valid exchange matrix: It is applied to uncompleted markets and the exchange is made among a subset of the completed sub-markets. The premises indicate that at least one completed market improves after the exchange and that none deteriorates. Let us remind that, in general, a market may generate both $M_i \xrightarrow{\overline{y}} M_i'$ and $M_i \overset{\overline{y}}{\longmapsto} M_i'$. So, the previous rule also considers situations where more than one market improves (we only require that at least one improves). Besides, let us remark that $M_i \xrightarrow{\overline{0}} M_i'$ always holds. So, a market not involved in the current exchange does not disallow the exchange. Regarding the conclusion, sub-markets belonging to M are modified according to the corresponding exchange matrix, while uncompleted sub-markets do not change.

For completeness reasons, some other minor rules concerning the propagation of the operator $\overset{\mathcal{E}}{\rightsquigarrow}$ must be added. Following the typical compositional style of process algebras, they show how the behavior of a part influences the behavior of the whole.

$$\frac{M_k \overset{\mathcal{E}}{\rightsquigarrow} M_k'}{\texttt{uncomp}(M_1, \ldots, M_k, \ldots, M_n) \overset{\mathcal{E}}{\rightsquigarrow} \texttt{uncomp}(M_1, \ldots, M_k', \ldots, M_n)}$$

$$\frac{M \overset{\mathcal{E}}{\rightsquigarrow} M'}{ms(M) \overset{\mathcal{E}}{\rightsquigarrow} ms(M')}$$

If a market reaches an optimum then we need to modify the attribute of the market by replacing a term such as $\texttt{uncomp}(M_1, \ldots, M_n)$ by a term such as $(id, S, T, u, \overline{x})$. Once a market is completed, resources are recursively moved from the corresponding agents to the leaves of the tree. Let us remark that, according to the scheme presented before and following [Núñez et al., 2005a], a market gets completed when all of its sub-markets are completed. However, in the following we will consider it in a different way. In particular, the rule defining how a market becomes a higher order market will differ from the rule given in [Núñez et al., 2005a]. As we said before, in order to allow the construction of a more practical design, the behavior originally described by the formal model is slightly modified in this paper. In particular, we adapt some aspects of the formal model to the feedback provided during the development of the system design. Concretely, we allow a market to become completed even if only a subset of agents is completed. Moreover, we allow the new higher order agent to be composed from only a subset of the agents it contains. By doing so, the formal model can also represent a situation where some agents are not considered because they try to join the new higher order market after the timeout was reached. Moreover, it can also represent an *alternative* approach where only those agents that do not increment their utility in the current level participate in the higher level. As we will see in the next section, this alternative will be motivated by efficiency issues during the development of the e-barter system in terms of web services.

These alternative behaviors were not be considered in the original formal model because they are motivated by low level details that are beyond the abstraction level of the specification. However, these issues turned out to be relevant to accurately describe the overall behavior of the e-barter system accordingly to the design presented in the next section. This is an example of the two-fold relation existing between the formal model and the web services oriented design, and how both can be used to adapt each other.

Note that the modification of the former formal model will consist, on the one hand, in enabling new behaviors that were not considered in the former model and, on the other hand, in removing some features that were not considered relevant for the implementation of the system by means of web services. Regarding the former, let us note that the model presented in [Núñez et al., 2005a] does not allow a market to become completed until all submarkets do so, and all submarkets are always included in the new agent. So, in this sense, the new specification can be regarded as a *generalization* of the previous one: According to the new specification, more behaviors are possible, so the former specification is a *refinement* of the new one (which, unfortunately, diverges from our current necessities). Regarding the latter, that is removing unnecessary functionalities from the model, let us note that the model in [Núñez et al., 2005a] allowed to explicitly consider the payment of *transaction* and *shipping* costs to the owner of the system. Concerning this, for the sake of simplicity we adopted the following design decision: Features devoted to representing the e-barter system as a

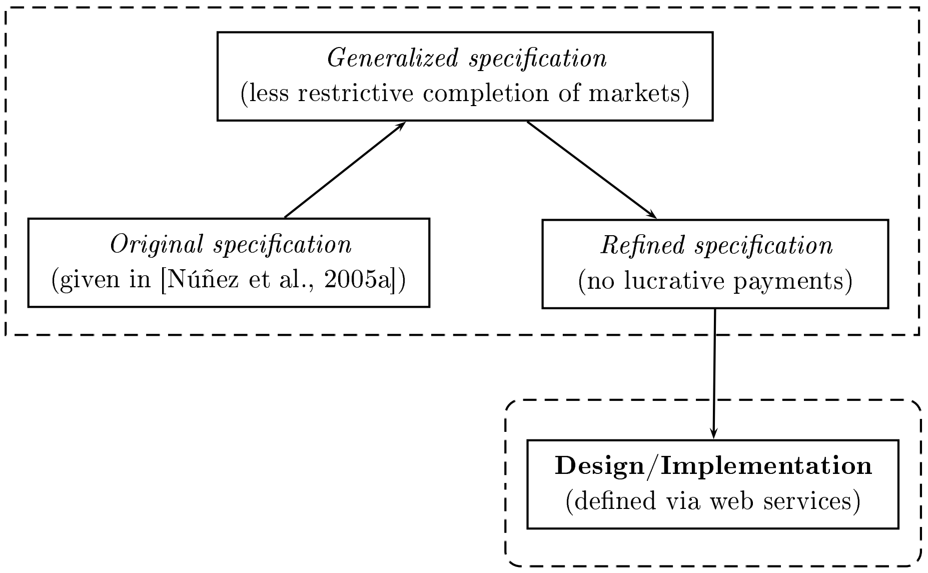
Generalized specification
(less restrictive completion of markets)
Original specification
(given in [Núñez et al., 2005a])
Refined specification
(no lucrative payments)
Design/Implementation
(defined via web services)

Figure 4: Generalization and refinement of the specification given in [Núñez et al., 2005a].

lucrative commercial activity for the owner of the system would not be regarded in the design. Hence, the previous costs were not considered. In this sense, the new model refines the previous one as described in [Núñez et al., 2005a]. In fact, the new model presented in this paper is created to enable the consistent construction of an implementation where some practical issues are dealt as we need. Note that the implementation is also a kind of (lower level) refinement. Thus, in this paper the model in [Núñez et al., 2005a] is both generalized and refined to allow a new (low level) refinement in a different direction. Figure 4 depicts this relation.

Let us present the rule allowing a market to become complete, which encapsulates the new capabilities commented before. The rule uses two auxiliary notions: `Deliver` and `CreateUtility`. Function `Deliver` distributes a basket of resources among the original agents that are located in the leaves of the tree that is provided. On the other hand, function `CreateUtility` computes a combined utility function from the ones provided as arguments as it is described in the algorithm shown in Section 2, step 3. These concepts will be formally defined afterwards in Definition 6. In the next definition, $A_1, \ldots, A_r$ represent the completed markets that will be represented by the new agent, while $B_1, \ldots, B_l$ are the completed/uncompleted markets that will not. Their indexes in the market are identified by I and I', respectively. In both cases, the resources obtained by completed markets (i.e. agents) must be delivered to the original agents. Hence, the `Deliver` function is applied in both cases to completed markets. Finally, let us note that the formal specification must give enough freedom to allow different alternative ways to choose which submarkets will be represented by the new agent. In order to do it, the following rule actually allows any combination, that is, it introduces non-determinism. The responsibility to fix a criterion to make this selection will be given to the design(s), as we will see in the following sections.

Definition 5. Let $M = \texttt{uncomp}(M_1, \ldots, M_n)$ be an uncompleted market. Let $I = \{s_1, \ldots, s_r\} \subseteq \{1, \ldots, n\}$ where for any $i \in I$ we have $M_i = (id_i, S_i, T_i, u_i, \overline{x}_i)$. Besides, let $I' = \{t_1, \ldots, t_l\} = \{1, \ldots, n\} - I$. The following rule modifies the market from uncompleted to completed:

$$\frac{M \not\rightsquigarrow}{M \rightsquigarrow (id, [A_1, \ldots, A_r], [B_1, \ldots, B_l], u, \sum_{i \in I} \overline{x_i})}$$

where

- id is a fresh identification symbol,
- $u = \texttt{CreateUtility}(u_{s_1}, \ldots, u_{s_r}, \overline{x_{s_1}}, \ldots, \overline{x_{s_n}})$,
- for all $1 \leq i \leq r$, $A_i = (id_{s_i}, S'_i, T_{s_i}, u_{s_i}, \overline{x_{s_i}})$ with $S'_i = \texttt{Deliver}(S_{s_i}, u_{s_i}, \overline{x_{s_i}})$,
- for all $1 \leq i \leq l$ we have that
  - if $M_{t_i} = (id_{t_i}, S_{t_i}, T_{t_i}, u_{t_i}, \overline{x_{t_i}})$ for some $id_{t_i}, S_{t_i}, T_{t_i}, u_{t_i}, \overline{x_{t_i}}$ (i.e., M_{t_i} is completed) then $B_i = (id_{t_i}, S'_i, T_{t_i}, u_{t_i}, \overline{x_{t_i}})$ and $S'_i = \texttt{Deliver}(S_{t_i}, u_{t_i}, \overline{x_{t_i}})$,
  - if $M_{t_i} = \texttt{uncomp}(M^1_{t_i}, \ldots, M^k_{t_i})$ for some $M^1_{t_i}, \ldots, M^k_{t_i}$ (i.e., M_i is uncompleted) then $B_i = M_{t_i}$.

□

The previous definition differs from the definition of the same rule given in [Núñez et al., 2005a] in that it does not require that all submarkets are completed. Instead, a set of completed markets (not necessarily involving *all* completed markets) is identified, and the new agent is created in such a way that only markets in this set are regarded.

Let us remark that in the previous rule the transition $\rightsquigarrow$ is not labelled. These transitions play a role similar to internal transitions in classical process algebras. In order to propagate the transformations given by a $\rightsquigarrow$ transition to the context of different constructors, other minor rules have to be added.

$$\frac{M_k \rightsquigarrow M'_k}{\texttt{uncomp}(M_1, \ldots, M_k, \ldots, M_n) \rightsquigarrow \texttt{uncomp}(M_1, \ldots, M'_k, \ldots, M_n)}$$

$$\frac{M \rightsquigarrow M'}{ms(M) \rightsquigarrow ms(M')}$$

The next definition formally presents some concepts used in Definition 5.

Definition 6. Let $A = (id, S, T, u, \overline{x})$ be an agent. The *allocation* of the basket of resources $\overline{x}$ among the agents belonging to S with respect to the utility function u, denoted by $\texttt{Deliver}(S, u, \overline{x})$, is recursively defined as:

$$\texttt{Deliver}(S, u, \overline{x}) = \begin{cases} [\,] & \text{if } S = [\,] \\ [M'_1, \ldots, M'_n] & \text{if } S = [M_1, \ldots, M_n] \end{cases}$$

where for any $1 \leq i \leq n$ we have that if $M_i = (id_i, S_i, T_i, u_i, \overline{x_i})$ then we have $M'_i = (id_i, \texttt{Deliver}(S_i, u_i, u(\overline{x}).i), T_i, u_i, u(\overline{x}).i)$.

Let us consider n pairs $(u_i, \overline{x_i})$. The *utility function* constructed from the utility functions $u_1, \ldots, u_n$ with respect to the baskets of resources $\overline{x_1}, \ldots, \overline{x_n}$, denoted by $\texttt{CreateUtility}(u_1, \ldots, u_n, \overline{x_1}, \ldots, \overline{x_n})$, is defined as:

$$\max\{(\textstyle\sum_i u_i(\overline{x'_i}), \overline{x'_1}, \ldots, \overline{x'_n}) \mid \textstyle\sum_i \overline{x'_i} = \overline{x} \ \wedge\ \forall\, 1 \leq i \leq n : u_i(\overline{x'_i}) \geq u_i(\overline{x_i})\}$$

We consider that the previous maximization is performed over the first argument (representing the *utility*) and we assume $\max \emptyset = (0, \overline{0}, \ldots, \overline{0})$. □

In [Núñez et al., 2005a] it is shown that if uncomplete markets become complete only after (a) all submarkets are completed and (b) no more fair exchanges are possible in the market then, when the top market is completed, the distribution of resources is *Pareto optimal*. This means that there does not exist an exchange among original agents such that, by performing it, the utility of at least one agent improves and no agent worsens. Let us note that this property is trivially met in a market where all agents are connected to a single market, because any fair exchange involving any group of agents is allowed in this market. However, it is more interesting if we consider other non-trivial structures of markets, because in these structures direct exchanges among original agents are not allowed. That is, in spite of the fact that several levels of *representative* agents are required to connect two distant agents in these structures, final distributions are Pareto optimal as well.

However, let us note that the rule given in this paper to turn an uncompleted market into a completed one (see Definition 5) requires neither (a) nor (b). As we commented before, these requirements were removed in this specification due to efficiency issues detected in the design given in the next section. Let us note that if these properties do not hold then the optimality of distributions is not met in general. We illustrate this with the following example.

Example 2. For $1 \leq i \leq 5$, let $A_i = (id_i, [\,], [\,], u_i, \overline{x_i})$ where

$$\begin{array}{ll} u_1(x,y,z) = 2x + y + 3z & \overline{x_1} = (0,1,0) \\ u_2(x,y,z) = x + 2y & \overline{x_2} = (1,0,0) \\ u_3(x,y,z) = 2x + 3y & \overline{x_3} = (0,0,0) \\ u_4(x,y,z) = 3x + z & \overline{x_4} = (0,0,1) \\ u_5(x,y,z) = 3x + 2z & \overline{x_5} = (0,0,0) \end{array}$$

Besides, let $M_1 = \texttt{uncomp}(A_1, A_2, A_3)$, $M_2 = \texttt{uncomp}(A_4, A_5)$, and $M_3 = \texttt{uncomp}(M_1, M_2)$. Finally, let $M = ms(M_3)$.

Let us suppose that agents A_1 and A_2 exchange one unit of good y by one unit of good x (within market M_1). This allows both agents to improve their utility. At this point, no more exchanges are allowed in M_1, so M_1 gets completed and becomes itself an agent A'_1. Let us suppose that, when a market becomes an agent, only those agents of the market that did not improve their utility are represented by the new agent in the upper level. According to this approach, only A_3 is represented by the new agent A'_1.

In parallel, M_2 gets completed as well (no exchange is possible within this market). Since neither A_4 nor A_5 improved their utility, both are represented by a new agent A'_2. According to this configuration of agents, no exchange will be possible in M_3: Since A'_1 owns nothing and A'_2 gives a positive utility to its unique belonging (one unit of good z), M_3 is trivially completed.

Next, goods are delivered top down to original agents in such a way that A_3, A_4, and A_5 remain the same (because no exchange was made in M_3). In addition, A_1 and A_2, previously removed from the structure of (active) markets, stay the same since they performed the exchange commented before. For $1 \leq i \leq n$, let $\overline{x'_i}$ represent the current basket of agent A_i. We have:

$$\begin{array}{l} \overline{x'_1} = (1,0,0) \\ \overline{x'_2} = (0,1,0) \\ \overline{x'_3} = (0,0,0) \\ \overline{x'_4} = (0,0,1) \\ \overline{x'_5} = (0,0,0) \end{array}$$

This distribution of goods is not optimal: If A_1 and A_3 exchange one unit of x by one unit of z then both agents improve. That is, the distribution obtained by the e-barter system under the assumption that only some agents are represented in upper levels is not Pareto optimal. □

In spite of the loss of optimality, we create utility functions of representative agents as it is defined in [Núñez et al., 2005a] because this provides an *ideal* limit of optimality. We know that, as long as all agents are actually completed and participate in higher order markets, the system tends to the optimality.

The last rules of the operational semantics are shown in Figure 5. They are used to *reset* the whole market structure when the top market is completed. The term σ denotes that a completed market must be reset, that is, turned back into uncompleted market. This symbol disappears when the market is actually reset, which might require to propagate it to the markets in the lower level. In fact, the rules shown in Figure 5 recursively traverse the market structure top down from the top market up to final agents denoting users. Resetting the system of markets prepares it for a possible new use. Moreover, when the reset process reaches basic agents (i.e., agents following the form $(id, [\ ], [\ ], u, \overline{x})$, thus representing end-users) agents may change their utility functions and their baskets of resources, according to their users preferences (see the second rule in Figure 5). A new execution of the system may be useful even if utility functions and baskets do not change. Let us note that if the final distribution of resources of the previous execution is Pareto optimal then, if utility functions do not change, it is pointless to execute the system one more time because no additional exchanges will be performed. However, since the rule denoting the completion of markets presented in Definition 5 does not guarantees the optimality of final distributions, a new execution might still improve the utility of some users. Let us note that, when a market is completed, the `Deliver` function is not applied to uncompleted markets below the market: Since these markets were not completed on time, the exchanges performed inside them were not made effective to final users. In this case, a *new* execution of the whole system of markets may be used to *continue* the evolution of these markets. In order to do so, the reset rules shown in Figure 5 do not reset uncompleted markets (see the fourth rule). In this way, the evolution of these markets can continue from the point they reached before on, in the next execution.

3 Design of multi-level e-barter Systems via web services

The specification defined in the previous section, which is a modification of the specification described in [López et al., 2002, López et al., 2003] and analyzed in [Núñez et al., 2005a], presents e-barter systems in terms of what can be done, for example, in terms of constraints on exchanges (they must satisfy the requirements imposed by utility functions). In the design phase, instead, we have to define the precise structure (architecture) of the system, the entities which play a role in it, the order in which things have to be done, and the temporal behaviour of such entities. For example, it has to be taken into account the location of the instances of such entities as well as which instances communicate with each other, the kind of interactions that the entities may perform, the kind of data exchanged, and the workflow of such interactions. Temporal issues are related to the system efficiency, which is important for a good design. For example, let us suppose that a market is temporaly out of service. If the higher level market is not allowed to complete until all submarkets get completed then the higher level will have to wait for a very long time until its submarket is recovered. It is preferable that the systems goes on (by using a timeout mechanism), ignoring

$$\frac{}{ms((id,S,T,u,\overline{x})) \hookrightarrow ms(\sigma((id,S,T,u,\overline{x})))}$$

$$\frac{u' \in \texttt{UtilFuncs} \ \wedge \ \overline{x}' \in \mathbb{R}^p_+}{\sigma((id,[\,],[\,],u,\overline{x})) \hookrightarrow (id,[\,],[\,],u',\overline{x}')}$$

$$\frac{S = [A_1,\ldots,A_n] \ \wedge \ T = [B_1,\ldots,B_m] \ \wedge \ n+m > 0}{\sigma((id,S,T,u,\overline{x})) \hookrightarrow \texttt{uncomp}(\sigma(A_1),\ldots,\sigma(A_n),\sigma(B_1),\ldots,\sigma(B_m))}$$

$$\frac{}{\sigma(\texttt{uncomp}(M_1,\ldots,M_n)) \hookrightarrow \texttt{uncomp}(M_1,\ldots,M_n)}$$

$$\frac{M_k \hookrightarrow M'_k}{\texttt{uncomp}(M_1,\ldots,M_k,\ldots,M_n) \hookrightarrow \texttt{uncomp}(M_1,\ldots,M'_k,\ldots,M_n)}$$

$$\frac{M \hookrightarrow M'}{ms(M) \hookrightarrow ms(M')}$$

Figure 5: Rules to reset a global market.

the damaged market by now. Afterwards, a new execution of the whole tree of markets will give this market a new chance to participate.

When producing the WS-BPEL description of the e-barter system the first thing to do is to identify the different entities (which in the WS-BPEL specification will be denoted by so-called portTypes) that are involved in the system. In particular, not only *concrete entities* (that will be actually designed via web-services) but also *abstract entities* which just correspond to human actions. Of course, abstract entities are assumed to be endowed with some electronic interface (e.g. a client PC) to interact with concrete entities (real web-services). In the case of the e-barter system we have three main entities: Clients, agents and markets. Each client belongs to a certain market of level 0 (the most local level in the hierarchy of markets). Agents in charge of making exchanges for clients belong to a certain market at level 0, while agents in charge of making exchanges for markets of level $n \geq 0$ belong to a certain market of level $n+1$. It also works the other way around, that is, if a particular market is located at level 0 then it determines a set of clients (we suppose that they are numbered from 1 on) and a set of agents.[3] On the contrary, if the market is located at a level $n > 0$ then it determines a set of agents only (numbered from 1 on) where each agent makes exchanges for a given market of level $n-1$.

[3] These agents are numbered as well and we suppose, for the sake of simplicity, that the agent i makes exchanges for client i.

In order to represent the architecture of a single market, the operations needed to manage the market have to be effectively provided by the system. What can be seen as a set of operations to manage the data involved in the market becomes an entity to support the behavior and evolution of the market. This entity is called the *manager of the exchange matrix.*

3.1 Behavior of the WS-BPEL design

The behavior of the three main entities is defined via orchestration with the semi-formal (XML based) workflow language WS-BPEL in such a way that is consistent with the formal model given in the previous section.

Conceptually, the design defined in WS-BPEL explicitly represents the communication between the entities, that in the formal model are simply included syntactically one inside each other. More precisely, according to the formal model, a market $\texttt{uncomp}(M_1, \ldots, M_n)$ that starts performing exchanges includes syntactically all the agents that trade in it, i.e. given $I = \{s_1, \ldots, s_r\} \subseteq \{1, \ldots, n\}$ such that $\{M_{s_1}, \ldots, M_{s_r}\}$ is the set of submarkets represented by trading agents, we have that, for all $i \in I$, M_i follows the agent form $(id_i, S_i, T_i, u_i, \overline{x_i})$. The corresponding market entity in the WS-BPEL design will first communicate with all the agent entities corresponding to $M_{s_1}, \ldots, M_{s_r}$ in order to get their utility functions $u_{s_1}, \ldots, u_{s_r}$ and baskets $\overline{x_{s_1}}, \ldots, \overline{x_{s_r}}$, and then it will perform exchanges according to the internal behavior of $\texttt{uncomp}(M_1, \ldots, M_n)$ in the formal model until it saturates. [4] When this happens, in the formal model we have that $\texttt{uncomp}(M_1, \ldots, M_n)$ can be turned into $A \equiv (id, [A_1, \ldots, A_r], [B_1, \ldots, B_l], u, \overline{x})$.

Supposing that we are not in the top level market, A represents the agent which is in charge for trading for this market in the upper level, where: $A_1, \ldots, A_r$ are the agents represented at the upper level by the new agent, $B_1, \ldots, B_l$ are all the other submarkets, and u and $\bar{x}$ are the aggregated utility function and the overall basket of resources. In the same way, in the design the market entity will simply transmit the same aggregated information to the agent entity (of the higher level) corresponding to A. The behaviour of the agent entity in the WS-BPEL design is defined in such a way that it checks the information received, and connects to its local market to deliver them and start trading (so the described flow repeats). In particular the execution of the transition that turns $\texttt{uncomp}(M_1, \ldots, M_n)$ into the agent A in the formal model represents the whole procedure above, i.e. also successful connection of the agent corresponding to A to its local market (in the design a market accepts connection from agents only before a given timeout, hence an attempt of connection can get stuck if such a timeout is expired).

[4] The WS-BPEL design abstracts from computation inside atomic services such as the evaluation of the exchanges to be made (the saturation of the matrix of exchanges). Therefore a legal implementation of such services must be made in such a way that it conforms with the formal model behavior.

On the contrary, if we are in the top level market, the execution of the transition that turns `uncomp`$(M_1, \ldots, M_n)$ into the agent A is just an abstract representation of the completion of the e-barter system and obviously does not correspond to a real activity of agent aggregation and communication with the upper level in the design.

In addition to the formal model, the WS-BPEL design includes also an explicit phase of transmission of results of the exchange performed to the clients (that in the formal design is just represented statically by applying the `Deliver` function to the syntactical tree of agents upon completion of markets). Such a transmission is started when the top-level market is completed and it is propagated top-down by markets and agents until the client entities are reached and notified. Note that, in the formal specification, the `Deliver` function is applied to agents that are aggregated for the upper level at the completion of each market (not just for the top-level market as in the design). This is just an abstract representation that is done for algebraic compositional reasons: Anyway top-down propagations of this kind at upper levels replace the previous propagations at lower levels. Therefore the design behavior is correct in this respect.

3.2 An alternative design choice

As already mentioned, we can also adopt an alternative design choice. In order to improve the efficiency of the system and to keep the structure of the utility functions (obtained by aggregation after the saturation of markets) as simple as possible, we can decide to propagate upwards only requests of agents who did not improve their utility in the current market. On the contrary, baskets of agents that improved their utility can be immediately distributed top-down to the clients. By following this strategy, the number of agents effectively participating in each market is reduced. This allows to reduce the time required to get all of them connected, as well as the time spent to perform exchanges upon completion.

Note that such an alternative design is still compliant with the formal specification: When `uncomp`$(M_1, \ldots, M_n)$ is turned into the agent A the formal model allows us to consider just a subset of the agents that traded in the market (and not necessarily all) for aggregation and deliver to the agent A of the upper level market. In particular in the formal representation of such an agent $A \equiv (id, [A_1, \ldots, A_r], [B_1, \ldots, B_l], u, \overline{x})$ we have that: $A_1, \ldots, A_r$ are the agents that are to be represented at the upper level (a subset of the agents that traded); $B_1, \ldots, B_l$ are all the other agents (to which the `Deliver` function is applied) and all the uncompleted markets in $M_1, \ldots, M_n$; and u and $\bar{x}$ are the aggregated utility function and the overall basket of resources. In the same way, in the alternative design the market entity will first deliver baskets to the agent entities corresponding to the agents (in $B_1, \ldots, B_l$) that will not participate in trading at the upper level (because they already improved their utility), then it will evaluate and transmit the aggregated information about remaining agents (in $A_1, \ldots, A_r$) to the agent entity of the higher level corresponding to A as described in the previous section.

As explained in Section 2, the price for the improved efficiency is the loss of global optimality of exchanges in general. In particular in Example 2 we showed a situation where the optimality is lost by following the behavior proposed in this section. More generally, the loss of optimality may be high in the worst case. For instance, if an agent performs only a tiny exchange at a certain market (e.g., 0.001 of good x by 0.001 of good y) that (slightly) improves its utility then, according to the alternative design, such agent will not be allowed to participate in the higher level market. In general, if the utility of some agents is only slightly improved inside a market then the improvement of the utility for the related users upon termination of the whole system will be negligible. Moreover, an agent that strongly increments its utility at a given market may also lose some utility because participating in higher markets could allow it to increase its utility even more.

However, there are several usual scenarios where the loss of generality may be very low. For example, let us suppose that the utility function of a user denotes that he is interested in exchanging a single unit of some good. Assume that he succeeds in this goal (e.g. because the only way for his market to reach completion is to perform the exchange of the whole unit he seeks). In this case, he has no reason to participate in the market anymore. In this case, blocking him from participating in higher levels does not lead to losing any optimality.

3.3 Local naming definition for each market

In this section we describe the mechanism to provide a uniform specification of the markets in our e-barter system by means of local naming and the technique based on UDDI to connect markets to agents of higher level and vice-versa (thus defining the structure of the system: This was done correspondingly by syntactical inclusion of markets in the formal model).

In order to invoke a service, other services must know where the first one is located. This mechanism is implemented through the binding of the names of the services to a physical address by using a UDDI server. In the case of our e-barter systems, following a hierarchical architecture, the structure of the UDDI service is depicted in Figure 6. In this graphic we can observe that there exists a different local UDDI server for every market.

Thus, we adopt the idea that each single market of the market system has its own *namespace*, defined just through a service of a UDDI. In each of these namespaces, every portType comes associated/bound to a specific web service. Thus, if one invokes a service appearing with the same name (same portType) in

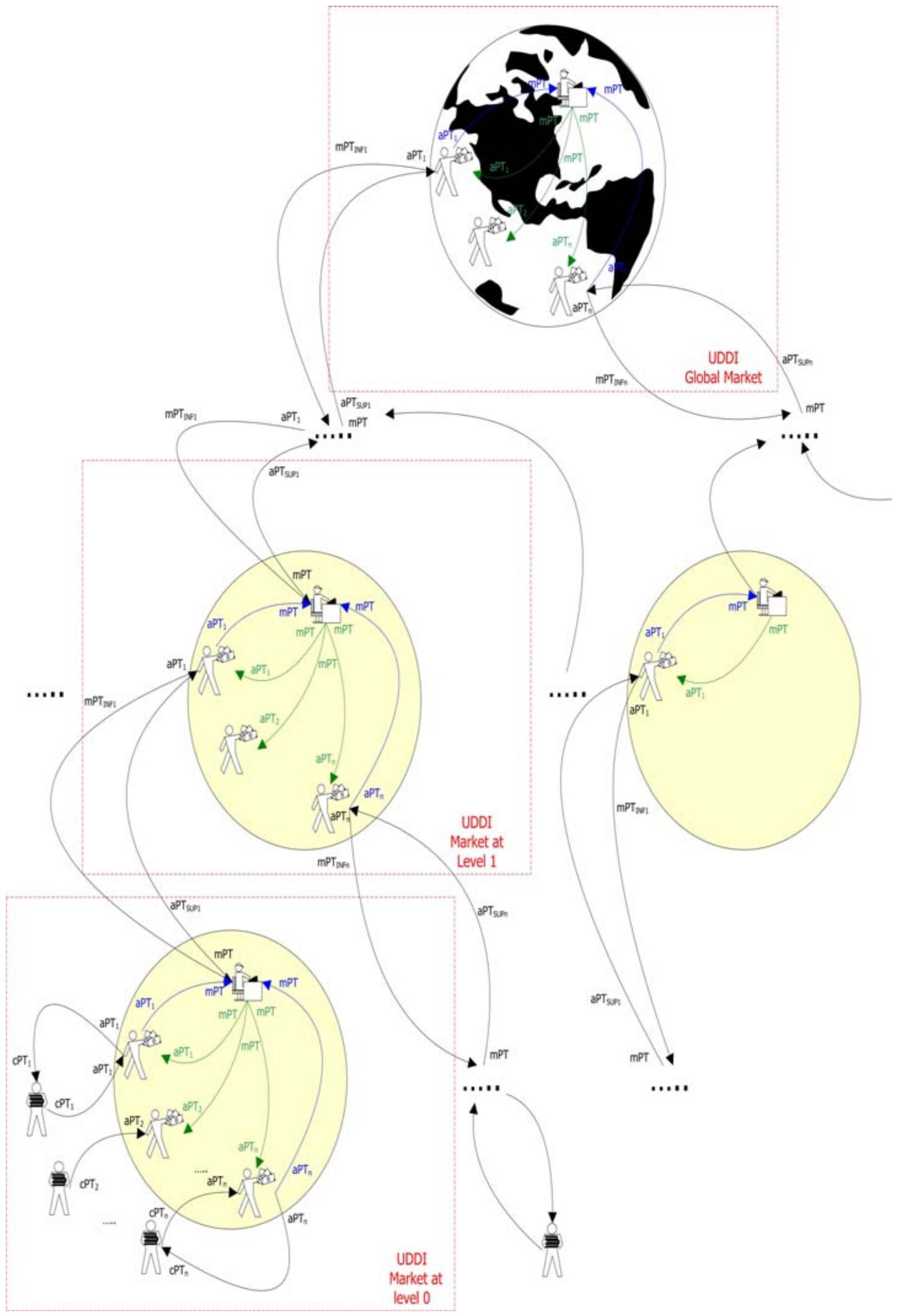

Figure 6: Scheme to bind names.

two different markets then the called service is the one determined by the UDDI associated with the market in which the invocation was performed. In this way, the specific hierarchical structure of e-barter systems can be uniform, that is, the same WS-BPEL process can be used for equal entities (regardless, e.g., of the level where they are located). The only difference is that the *client* portType is available only in the first level since instances of this type can communicate only with agents located in the first level.

If we are working in a market at a certain level, it is possible to refer to the agent $\mathrm{aPT_i}$ of the associated market at the immediately higher level by using the $\mathrm{aPT_{SUPi}}$ portType (it is assumed that the UDDI of the market is defined in such a way that $\mathrm{aPT_{SUPi}}$ binds to the i-th agent of the market at the higher level). Similarly, given a market at a certain non-zero level, it is possible to refer to the the market mPT at the immediately lower level for which the i-th agent of the market is in charge, by using the $\mathrm{mPT_{INFi}}$ portType.

4 Web Service design yielding global optimality

In this section we present a web-service design of the e-barter system formally specified in section 2. This design is developed in such a way that, supposing that for each market all agents connect to the market before the timeout, global optimatility of exchanges is guaranteed for the clients of the e-barter system.

4.1 WSDL definition of ports and partner links

In Figure 7 we present the WSDL graphical notation that we will use in this paper. In the following, we describe the meaning of port-types (i.e. types of services) that we use and their operations. In general names of operations are quite self-explaining: The name of the operation expresses the meaning of the action performed from the viewpoint of the port-type that offers it (not from the viewpoint of the user of the operation). As we already explained, port-types are used in our design to represent both "concrete" entities, i.e. real services that are bound at some location in the internet and "abstract" entities, i.e. the behaviour that a human must have to play this role and interact successfully with the real services. Port-types corresponding to abstract entities are listed in Figure 9, while concrete port-types are listed in Figure 10. Partnerlink-types are listed in Figure 8. Parnerlink-types in WS-BPEL express the kind of interactions that port-types may have: A pair denotes a kind of interaction where the port-types interact by invoking each other, a standalone port-type denotes a kind of interaction where such a port-type is just invoked by others.

In the following we give a brief explanation of concrete and abstract port-types and their operations. A more precise meaning and behavior of operations will be made clear when we will describe workflows in the next section.

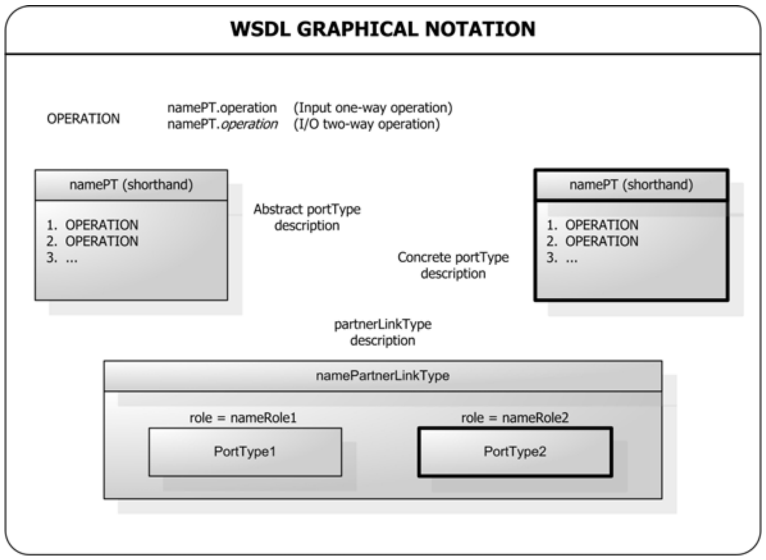

Figure 7: WSDL graphical notation.

- clientPT$_\mathrm{i}$, agentPT$_\mathrm{i}$ and marketPT correspond to the three main entities whose behavior is specified with WS-BPEL in the following.
- agentPT$_\mathrm{SUPi}$ is a portType identical to agentPT$_\mathrm{i}$. It contains the same operations of a the agent agentPT$_\mathrm{i}$ and it has the same behavior. The only difference is that it is bound by the UDDI service of a given market to a different location: The location of the agent agentPT$_\mathrm{i}$ (in the immediately higher level market) associated to such a market.
- marketPT$_\mathrm{INFi}$ is, similarly, a portType identical to marketPT. It contains the same operations of a market marketPT and it has the same behavior. The only difference is that it is bound by the UDDI service of a given market to a different location: The location of the market marketPT (in the immediately lower level) associated to the i-th agent of such a market.
- actionsClientPT$_\mathrm{i}$ represents the decisions that a customer takes.
- actionsAgentPT$_\mathrm{i}$ is a concrete entity representing an electronic support service for agents. In particular it contains a concrete service for controlling the validity of utility functions.
- actionsMarket is a concrete entity representing actions of markets not directly representable in the WS-BPEL workflow for markets. It contains: A

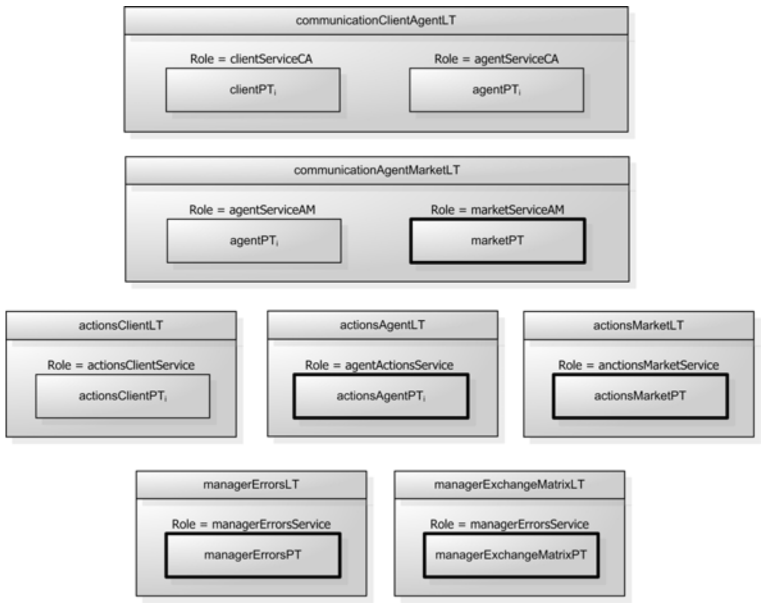

Figure 8: List of Partner-link Types.

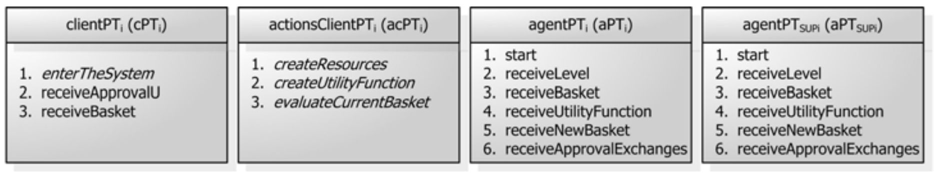

Figure 9: List of abstract portTypes.

service that returns the index i of the agent in charge of the current market in the higher level (0 if the current market is the top-level market) and a service that evaluates the next time deadline for connection of agents.

– managerErrorsPT is an entity used for notification of errors.

– managerExchangeMatrixPT is the entity that offers more services to the system. It is used to store the matrix of exchanges for a given market (and related information) and to manage it in an efficient way. It includes operation for the updating, modification and reading of such a matrix. In particular, *saturateMarket* saturates the matrix evaluating an optimal solution in an unspecified way; *maximizeU* and *sumBaskets* compute the aggregated

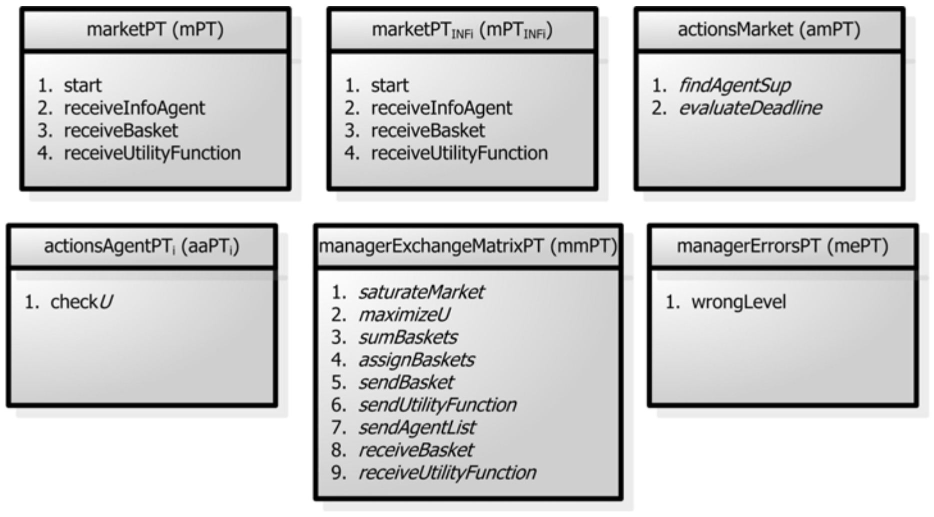

Figure 10: List of concrete portTypes.

utility function and the union of baskets after saturation, respectively; *assignBaskets* distributes an updated basket of resources received from the higher level to the agents trading in this market; and *sendAgentList* and *sendBasket* return an array with the list of indexes of agents trading in the current market and return the basket of resources assigned to the specified agent, respectively.

Note that the classification of the port-types above into concrete and abstract ones is based on the following assumptions. Clients and their actions are assumed to be just the representation of human behaviors. Agents are assumed to be human but we assume that they perform their actions in a computer-supported way (i.e. by means of a concrete service). Markets are assumed to be totally electronic.

4.2 WS-BPEL workflows for the entities of the e-barter system

In this section we present the WS-BPEL workflows defining the behavior of the *clients*, *agents*, and *markets* portTypes. In Figure 11 we present the WS-BPEL graphical notation that we will use in this paper. For the sake of simplicity we do not include the semi-formal behavioral representation in terms of XML code for each of the entities, but we give a graphical representation of the workflows (which is correspondent to the XML and more readable than the XML code). To be precise, in the workflow representation we assume invoke statements that do

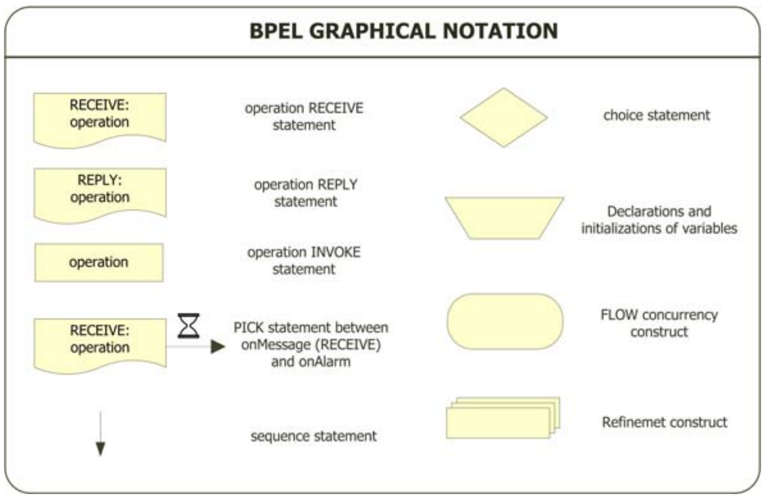
BPEL GRAPHICAL NOTATION
RECEIVE: operation
operation RECEIVE statement
REPLY: operation
operation REPLY statement
operation
operation INVOKE statement
RECEIVE: operation
PICK statement between onMessage (RECEIVE) and onAlarm
sequence statement
choice statement
Declarations and initializations of variables
FLOW concurrency construct
Refinemet construct

Figure 11: BPEL graphical notation.

not succeed (because the invoked web-service is not waiting in the corresponding receive statement) to keep trying, possibly via a timeout-based exception handling mechanism, until the invoke is successful. In this graphical representation, we will also assume that usual basic types of programming languages are mapped into corresponding XML Schema type descriptions (e.g. we make use of arrays for variables).

Finally, in the presentation of the workflows we use a refinement construct which allows us to express an entire workflow as a single box in a larger workflow including it. Such a construct simply stands for replacement of the box with the content of the refinement and is used for clarity of presentation.

4.3 Business process for clients

The workflow that represents the (human) behavior of the client identified as i-th in the namespace of its local (0-level) market is represented in Figure 12. The interaction of the client with the e-barter system (to exchange goods with other clients that may be located anywhere else in the world and to obtain in this way a satisfactory new basket of resources) is abstractly represented by the operation "enterTheSystem". First of all he contacts its corresponding i-th agent in the local namespace and he communicates him that he is working at level 0 (this is needed for technical reasons to maintain the description of the agent's behavior independent of the namespace). Then he performs the refined workflow presented in the lefthand side of Figure 13: He builds a basket of resources (pos-

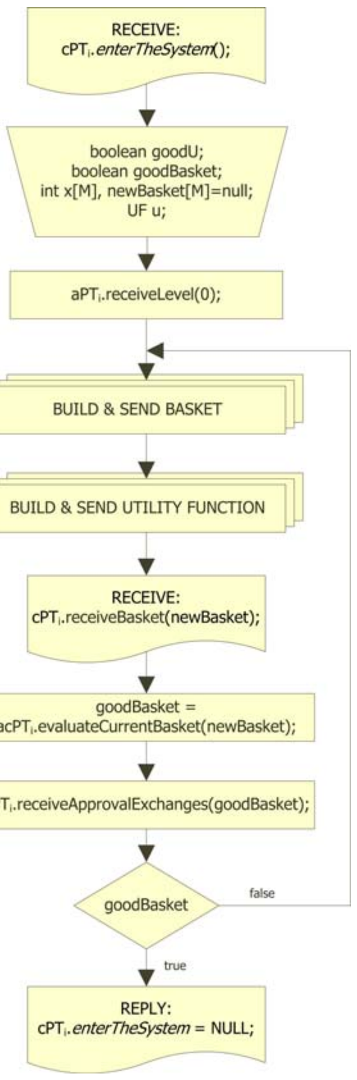
RECEIVE:
cPTi.enterTheSystem();
boolean goodU;
boolean goodBasket;
int x[M], newBasket[M]=null;
UF u;
aPTi.receiveLevel(0);
BUILD & SEND BASKET
BUILD & SEND UTILITY FUNCTION
RECEIVE:
cPTi.receiveBasket(newBasket);
goodBasket =
acPTi.evaluateCurrentBasket(newBasket);
aPTi.receiveApprovalExchanges(goodBasket);
goodBasket
false
true
REPLY:
cPTi.enterTheSystem = NULL;

Figure 12: Workflow of the i-th client (port-type "clientPT$_i$")

sibly determined on the basis of the "newBasket" basket of resources obtained from a previous interaction with the e-barter system) to be entered into the e-barter system via an abstract call to the operation "createResources" of the "actionsClient" port-type and sends it to the agent via its "receiveBasket" operation. Similarly, as defined by the workflow in the righthand side of Figure 13 he builds a utility function and he sends it to the agent: The created utility

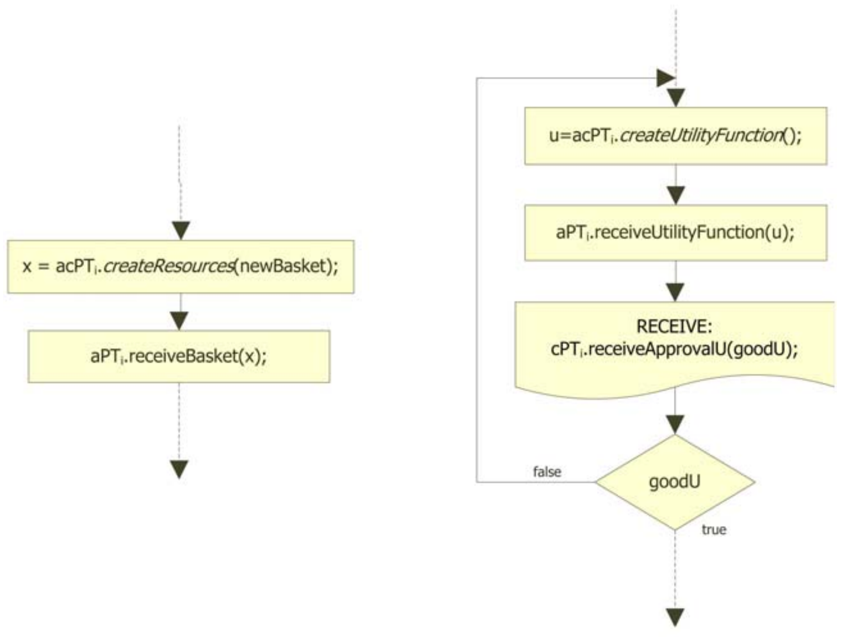
x = acPTi.createResources(newBasket);
aPTi.receiveBasket(x);
u=acPTi.createUtilityFunction();
aPTi.receiveUtilityFunction(u);
RECEIVE:
cPTi.receiveApprovalU(goodU);
false
goodU
true

Figure 13: Refinements of the client workflow: Refinement of *Build & Send Basket* (left) and refinement of *Build & Send Utility Function* (right).

function is subject to approval by the agent (who must give an approval verdict to the client by invoking its “receiveApprovalU” operation), hence the client must cycle until approval is positive. Once this happens, the client just waits for the outcome of the e-barter system execution: A new basket of resources that the agent sends to the client by invoking its “receiveBasket” operation. The received basket is then evaluated (operation “evaluateCurrentBasket” of the abstract “actionsClient” port-type) and the verdict is sent to the agent by invoking its “receiveApprovalExchanges” operation. If the verdict is not positive the interaction with the agent continues in a cycling way: Another basket of resources to be entered into the e-barter system is built.

4.4 Business process for agents

The workflow that represents the (human) behavior of the agent identified as i-th in the namespace of its associated market is represented in Figure 14. The initial event (that happens once and for all) of the agent becoming operational is abstractly represented by the operation “start”. The agent may directly be a representative of a client of the e-barter system or of a lower-level market de-

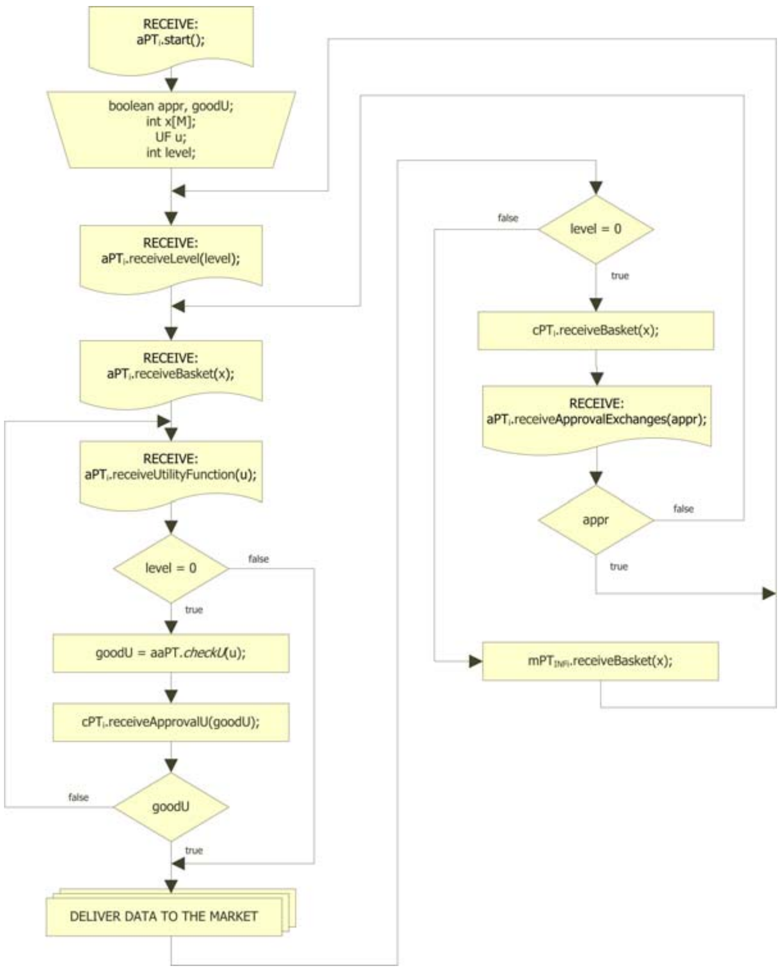
RECEIVE: aPTi.start();
boolean appr, goodU; int x[M]; UF u; int level;
RECEIVE: aPTi.receiveLevel(level);
RECEIVE: aPTi.receiveBasket(x);
RECEIVE: aPTi.receiveUtilityFunction(u);
level = 0
false
true
goodU = aaPT.checkU(u);
cPTi.receiveApprovalU(goodU);
goodU
false
true
DELIVER DATA TO THE MARKET
level = 0
false
true
cPTi.receiveBasket(x);
RECEIVE: aPTi.receiveApprovalExchanges(appr);
appr
false
true
mPT[Nf].receiveBasket(x);

Figure 14: Workflow of the i-th agent (port-type "agentPT$_i$").

pending on the level of its namespace (i.e. of its associated market). He initially needs to be informed about the level at which he operates: He expects his operation "receiveLevel" to be invoked (by the client or the lower level market). Then he expects to receive a basket of resources (via its operations "receiveBasket" and "receiveUtilityFunction") and a utility function: He will use them to participate in the exchanges with other agents in its associated market. In the case the level is "0", i.e. he is directly operating on behalf of a client, the utilityFunction is checked for consistency by invoking the "check" operation of the "actionsAgentPT$_i$" port-type and the verdict is sent to the client. The refined

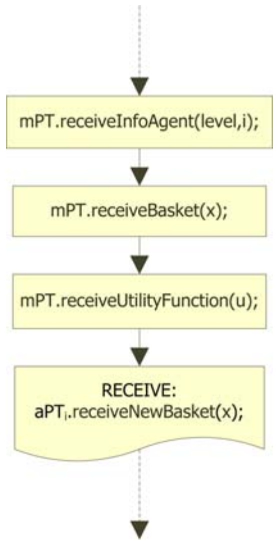
mPT.receiveInfoAgent(level,i);
mPT.receiveBasket(x);
mPT.receiveUtilityFunction(u);
RECEIVE:
aPT$_i$.receiveNewBasket(x);

Figure 15: Refinements of the agent entity: Refinement of *Deliver Data to the Market*.

workflow presented in Figure 15 defines how the above collected data is delivered to the market by invoking the operations of the market “receiveInfoAgent” (which provides information about the identity “i” of the agent and the level at which operates), “receiveBasket” and “receiveUtilityFunction”. Moreover according to the refined workflow, the agent waits for the outcome of the exchange activity at the associated market (which also involves performing exchanges at upper level markets): A new basket of resources that the market sends to the agent by invoking its “receiveNewBasket” operation. In the case the agent is operating at level 0, such a basket is then delivered to the client and an approval verdict is expected (operation “receiveAprrovalExchanges” of the agent): If the verdict is negative a new basket will be received from the client and the behaviour above is repeated. Similarly, if the agent is instead operating at upper levels, it will deliver the basket to the associated lower level market (port-type “marketPT$_{INF_i}$”).

4.5 Business process for markets

The workflow that represents the behavior of a market is represented in Figure 16. The initial event (that happens once and for all) of the market becoming operational is abstractly represented by the operation “start”. The system first establishes a timeout deadline for the connection of agents (operation “evalu-

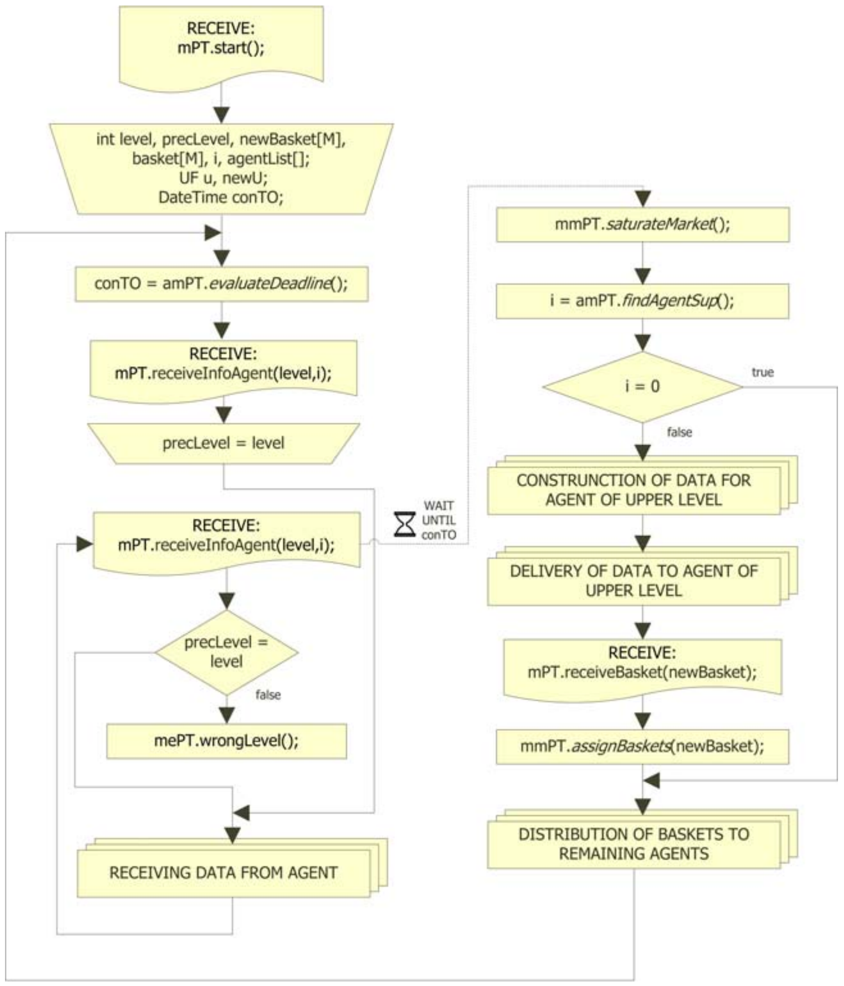
RECEIVE: mPT.start();
int level, precLevel, newBasket[M], basket[M], i, agentList[]; UF u, newU; DateTime conTO;
conTO = amPT.evaluateDeadline();
RECEIVE: mPT.receiveInfoAgent(level,i);
precLevel = level
RECEIVE: mPT.receiveInfoAgent(level,i);
precLevel = level
false
mePT.wrongLevel();
RECEIVING DATA FROM AGENT
WAIT UNTIL conTO
mmPT.saturateMarket();
i = amPT.findAgentSup();
i = 0
true
false
CONSTRUNCTION OF DATA FOR AGENT OF UPPER LEVEL
DELIVERY OF DATA TO AGENT OF UPPER LEVEL
RECEIVE: mPT.receiveBasket(newBasket);
mmPT.assignBaskets(newBasket);
DISTRIBUTION OF BASKETS TO REMAINING AGENTS

Figure 16: Workflow of the market (port-type "marketPT")

ateDeadline" of the port-type "actionsMarket") and stores it in the "conTO" variable. Then it waits for the connection of at least one agent: Its data is used to establish the level at which the market operates and every agent connecting afterwards is expected to consistently communicate the same level (if this does not happen the system stops by invoking the "wrongLevel" operation of the "managerErrors" port-type indicating a wrong configuration of the UDDI servers of the e-barter system). Every time a new agent connects the refined workflow presented in the righthand part of Figure 18 is executed. A basket of resources and a utility function is received from the agent (via the operations "receive-

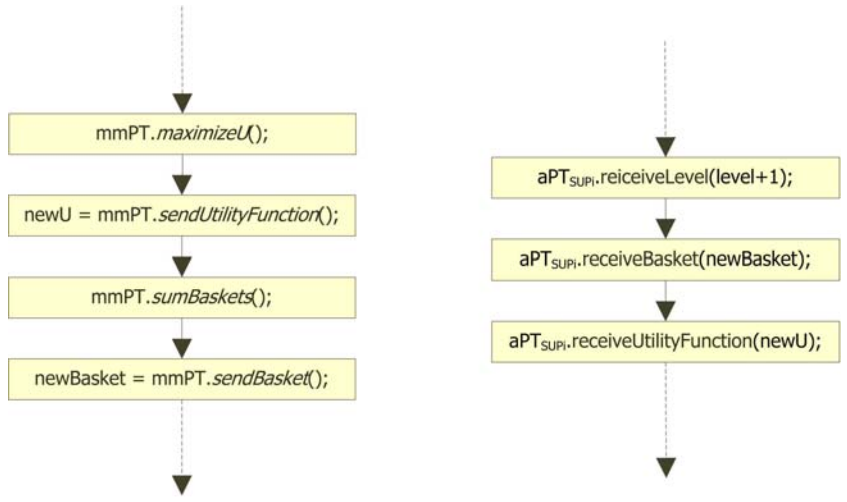
mmPT.maximizeU();
newU = mmPT.sendUtilityFunction();
mmPT.sumBaskets();
newBasket = mmPT.sendBasket();
aPTSUPi.reiceiveLevel(level+1);
aPTSUPi.receiveBasket(newBasket);
aPTSUPi.receiveUtilityFunction(newU);

Figure 17: Refinements of the market entity: Refinement of *Construction of Data for Agent of Upper Level* (left) and refinement of *Delivery of Data to Agent of Upper Level* (right).

Basket" and "receiveUtilityFunction" of the market): Both information are then forwarded (together with the identity "i" of the agent) to the "managerEchange-Matrix" entity by invoking its 'receiveBasket" and "receiveUtilityFunction" operations. Connections from agents are accepted until the time deadline "conTO" is reached. When such a time deadline is reached the market (possibly after the amount of time needed for finishing the management of a previously connected agent) executes the workflow in the righthand side of Figure 16. First of all the market is "saturated" (goods are exchanged between agents until no further exchange is possible) by invoking the operation "saturateMarket" of the "managerEchangeMatrix" entity. Then the operation "findAgentSup" of the port-type "actionsMarket" is invoked to determine the index i of the agent that represents the current market at the upper level. This operation is assumed to just return the special value "0" in the namespace of the top level market. In the case the market is not at the top level it behaves as follows. The lefthand part of the refined workflow presented in Figure 17 is performed: We invoke operations of the "managerEchangeMatrix" to compute and receive the aggregated utility function (which maximizes individual utility functions) and to compute and receive the aggregated basket of resources (the "sum" of all individual baskets). Once the data for the agent "agentPT$_{SUP_i}$" of the upper level is constructed in this way, the refined workflow presented in the righthand part of Figure 18 performs the actual delivery: The agent is informed he will work at one level

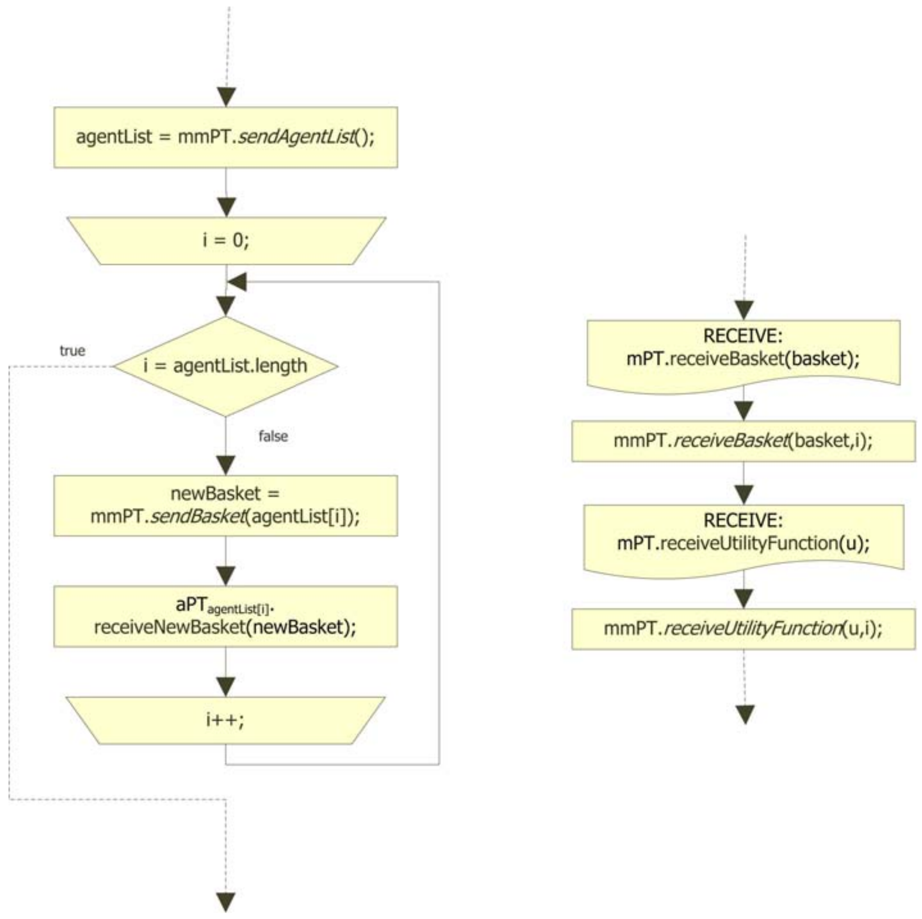
agentList = mmPT.sendAgentList();
i = 0;
true
i = agentList.length
false
newBasket = mmPT.sendBasket(agentList[i]);
aPT agentList[i]. receiveNewBasket(newBasket);
i++;
RECEIVE: mPT.receiveBasket(basket);
mmPT.receiveBasket(basket,i);
RECEIVE: mPT.receiveUtilityFunction(u);
mmPT.receiveUtilityFunction(u,i);

Figure 18: Refinements of the market process: Refinement of *Distribution of Baskets to Agents* (left) and refinement of *Receiving data from Agent* (right).

more than the level of the market and is assigned the aggregated basket and utility function. The market then waits for the outcome of the e-barter system at the upper levels: A new basket of resources that the agent sends to the market by invoking its “receiveBasket” operation. The received basket is then redistributed among agents registered at the market (operation “assignBasket” of the “managerEchangeMatrix” port-type). The workflow then continues in the same way no matter if the market is at the top level or not. The individual baskets are delivered to the agents by executing the refined workflow presented in the lefthand part of Figure 18. First of all this refined worflow receives from the “managerEchangeMatrix” an array containing all indexes of registered agents.

Then for every index (identifying an agent in the namespace of the market) recorded in such an array, the individual basket for the agent is received from the "managerEchangeMatrix" and is delivered to the agent.

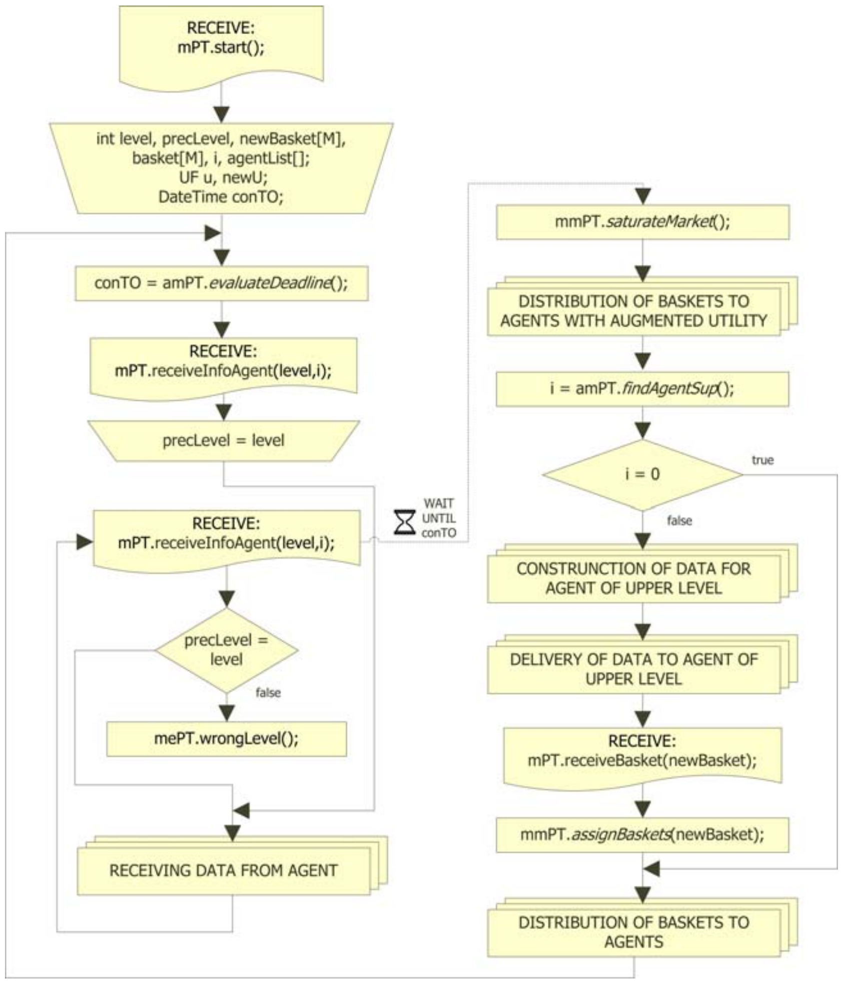
RECEIVE:
mPT.start();
int level, precLevel, newBasket[M], basket[M], i, agentList[]; UF u, newU; DateTime conTO;
conTO = amPT.evaluateDeadline();
RECEIVE:
mPT.receiveInfoAgent(level,i);
precLevel = level
RECEIVE:
mPT.receiveInfoAgent(level,i);
precLevel = level
false
mePT.wrongLevel();
RECEIVING DATA FROM AGENT
WAIT UNTIL conTO
mmPT.saturateMarket();
DISTRIBUTION OF BASKETS TO AGENTS WITH AUGMENTED UTILITY
i = amPT.findAgentSup();
i = 0
true
false
CONSTRUNCTION OF DATA FOR AGENT OF UPPER LEVEL
DELIVERY OF DATA TO AGENT OF UPPER LEVEL
RECEIVE:
mPT.receiveBasket(newBasket);
mmPT.assignBaskets(newBasket);
DISTRIBUTION OF BASKETS TO AGENTS

Figure 19: Alternative workflow of the market (port-type "marketPT")

5 Alternative web service design

In this section we present an alternative design that still complies with the formal specification we have seen in Sect. 2. The idea is to improve the efficiency of the

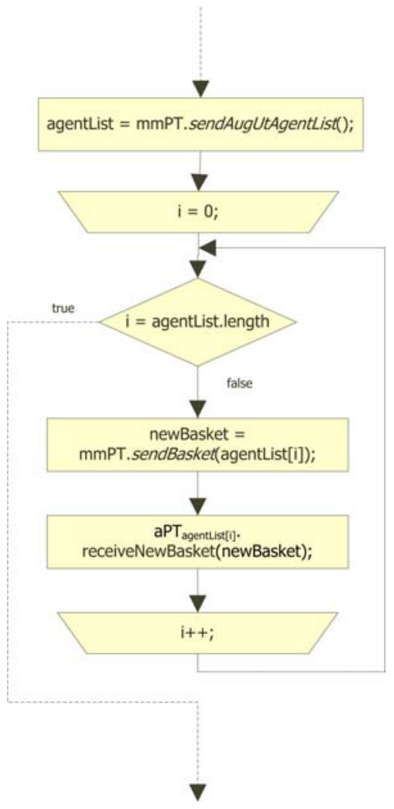
agentList = mmPT.sendAugUtAgentList();
i = 0;
true
i = agentList.length
false
newBasket = mmPT.sendBasket(agentList[i]);
aPT agentList[i].receiveNewBasket(newBasket);
i++;

Figure 20: Refinements of the alternative market entity: Refinement of *Distribution of Baskets to Agents with Augmented Utility.*

system and to keep the structure of the aggregated utility functions (obtained by aggregation after the saturation of markets) as simple as possible at the price of loosing global optimality of the exchanges in general. This is done by modifying the behaviour of markets in such a way that they propagate upwards only requests of agents who did not improve their utility after the exchanges. On the contrary, baskets of agents that improved their utility are immediately distributed top-down to the clients. Here the idea is that, if a client is not satisfied by the exchanges performed, he can re-enter the system so to try to get a further improvement, if possible.

The new design is obtained from the one presented in the previous section by: (*i*) adding the operation "sendAugUtAgentList" to the "managerExchange-MatrixPT" port-type, that returns an array with the indexes of agents that augmented their utility functions during exchanges; (*ii*) assuming that the old operations “sendAgentList”, “maximizeU”, “sendUtilityFunction”, “sumBaskets” and “sendBasket” of the same port-type now just deal with agents that did not augment their utility functions during exchanges; and (*iii*) by modifying the behavior of markets as shown in Figure 19. With respect to the previous workflow in Figure 16, now the market immediately distributes the baskets to the agents that augmented their utility functions during exchanges (before creating and delivering data to the upper level) by means of the refined workflow “Distribution of Baskets to Agents with Augmented Utility” presented in Figure 20. Its behavior is similar as that of the “Distribution of Baskets to Agents” refined workflow (previously presented in the lefthand part of Figure 18): The only difference is that the array containing agent indexes is receives from the “managerEchange-Matrix” by means of the new “sendAugUtAgentList” operation instead of the “sendAgentList” operation.

6 Conclusions

An e-barter system is an e-commerce environment where agents exchange resources on behalf of their respective users, exchanges do not necessarily involve money, and agents and markets are structured in a hierarchical fashion in such a way that markets may become higher order (representative) agents. This kind of systems were formally specified in [López et al., 2002, López et al., 2003]. Unfortunately, there is a gap between the formal specification and a suitable design for the system. While the formal specification level turns out to be fundamental in defining the functional ideal behavior of the system, several practical issues were not addressed in the formal specification. In this paper we have developed two alternative designs of an e-barter system in terms of *web services*. Web services provide a suitable and well-supported model (a number of related standard and tools have been developed) for defining the behavior of e-barter systems in a distributed way. Tasks like conceptual decomposition and parallel execution of independent activities by means of different entities are implicitly performed as a result of the definition of the system as a set of web service orchestrations. In doing this, the formal specification represents the ideal behavior of the system which is to be realized by adopting adequate design (architectural) choices at the level of Web Service orchestration. By developing this case study we experienced: Need of local naming technique via multiple UDDI services, introduction of new entities (such as the manager of the exchange matrix), design decisions about synchronization of re-start of cycles of exchanges in different markets, design decisions in structure of data exchanged and consequent tradeoff between optimality of exchanges and efficiency of the system, etc. Besides, let us note that a definition of the system in terms of web services does not only provide a suitable design, but also (partially) an implementation, since web service orchestrations defined with WS-BPEL are executable by ad-hoc interpreters.

We have also explored how a formal specification and a web services design can influence each other. Though the influence of the specification in the design is very clear, the way in which the design allows to refine the specification itself is remarkable. In fact, it allowed us to address one of the main problems of formal methods: The loss of information due to the abstraction. Since models must be simple and usable, they must abstract some details, but these details may turn out to be relevant to the specification afterwards (even if they are *still* abstracted). For example, in spite of the fact that our process algebraic specification does not represent the passing of time (time is *abstracted* in it), temporal issues detected during the development of the design motivated a change in the specification: Since agents can be delayed *in the design*, it is not feasible that all agents are connected on time. Thus, the specification rule for completing a market was changed to permit that not all agents connect to the market. That is, a detail that is abstracted in the specification (the time) motivates a change in another detail that is not (the completion of markets). Thus, the development of a design may be critical to produce a useful model.

Acknowledgements

Research partially funded by the EU Integrated Project Sensoria (contract number 016004), by the Spanish Ministerio de Educación y Ciencia projects MASTER/TERMAS (TIC2003-07848-C02-01) and WEST/FAST (TIN2006-15578-C02-01), the Junta de Castilla-La Mancha project PAC06-0008-6995, and the Marie Curie Research and Training Network *TAROT* (MRTN-CT-2004-505121).

References

[Andrews and Curbera, 2004] Andrews, T. and Curbera, F. (2004). "Web Service Business Process Execution Language, Working Draft". Version 2.0, 1.

[Bravetti et al., 2006] Bravetti, M.; Casalboni, A.; Núñez, M.; and Rodríguez, I. (2006). "From theoretical e-barter models to an implementation based on web services". In *IPM Int. Workshop on Foundations of Software Engineering (FSEN'05)*, pages 241–264. Electronic Notes in Theoretical Computer Science 159, Elsevier.

[Cavalli and Maag, 2004] Cavalli, A. and Maag, S. (2004). "Automated Test Scenarios Generation for an e-barter System". In *19th ACM Symposium on Applied Computing, SAC'04*, pages 795–799. ACM Press.

[Christenses et al., 2001] Christenses, E.; Curbera, F.; Meredith, G.; and Weerawarana, S. (2001). "Web Services Description Language (WSDL 1.1)". Note 15, `http://www.w3.org/TR/wsdl`.

[Hindriks et al., 1998] Hindriks, K.; de Boer, F.; van der Hoek, W.; and Meyer, J.-J. (1998). "Formal Semantics for an Abstract Agent Programming Language". In *Intelligent Agents IV, LNAI 1365*, pages 215–229. Springer.

[Huhns and Singh, 2005] Huhns, M. and Singh, M. (2005). "Service-Oriented Computing: Key Concepts and Principles". In *IEEE Internet Computing*, pages 75–81. IEEE Computer Society Press.

[López et al., 2002] López, N.; Núñez, M.; Rodríguez, I.; and Rubio, F. (2002). "A Formal Framework for e-barter based on Microeconomic Theory and Process Algebras". In *Innovative Internet Computer Systems, LNCS 2346*, pages 217–228. Springer.

[López et al., 2003] López, N.; Núñez, M.; Rodríguez, I.; and Rubio, F. (2003). "A Multi-Agent System for e-barter including Transaction and Shipping Costs". In *18th ACM Symposium on Applied Computing, SAC'03*, pages 587–594. ACM Press.

[Milner, 1989] Milner, R. (1989). *Communication and Concurrency.* Prentice Hall.

[Núñez et al., 2005a] Núñez, M.; Rodríguez, I.; and Rubio, F. (2005a). "Formal Specification of multi-agent e-barter systems". *Science of Computer Programming (to appear).* in press.

[Núñez et al., 2005b] Núñez, M.; Rodríguez, I.; and Rubio, F. (2005b). "Specification and Testing of Autonomous Agents in e-commerce systems". *Software Testing, Verification and Reliability*, 15(4). in press.

[Probert et al., 2003] Probert, R.; Chen, Y.; Ghazizadeh, B.; Sims, P.; and Cappa, M. (2003). "Formal Verification and Validation for e-commerce: Theory and best Practices". *Journal of Information and Software Technology*, 45(11), pp. 763–777.

[Rao, 1996] Rao, A. (1996). "AgentSpeak(L): BDI agents speak out in a logical computable language". In *Agents Breaking Away, LNAI 1038*, pages 42–55. Springer.

[SOAP, 2006] SOAP (2006). "Simple Object Access Protocol". `http://www.w3.org/TR/soap`.

[UDDI, 2006] UDDI (2006). "Universal Description, Discovery and Integration of Web Services". `http://www.uddi.org/specification.html`.

[W3C, 2006] W3C (2006). "World Wide Web Consortium". `http://www.w3.org`.