Autonomic tracing of production processes with mobile and agent-based computing

Mario G.C.A. Cimino *, Francesco Marcelloni

Dipartimento di Ingegneria dell'Informazione: Elettronica, Informatica, Telecomunicazioni, University of Pisa, Via Diotisalvi 2, 56122 Pisa, Italy

Keywords:
Traceability
Mobile information systems
Business process modeling
Self-organizing system
Context-aware user interface

A B S T R A C T

Tracing items in a supply chain, across different enterprises and through the full processes scope, is today an inherently complex design task. Enterprises are typically comprised of hundreds of applications that are custom built at different times, acquired from third parties and parts of legacy systems, and also operating in multiple tiers of different manufacturing and information system platforms. Further, traceability is characterized by a goal-oriented approach, in which business-process analyses are driven by goal achievements rather than by systematic engineering processes. The use of a classical enterprise integration approach mostly needs tailoring to different applications. Due to the number and diversity of the systems and of their interactions, and to their dynamicity, it is difficult, costly, and therefore often not convenient to develop in large scale distributed systems.

To overcome these issues, a supply chain traceability system with a high level of automation is discussed in this paper. In particular, the system adopts an agent-based approach, in which cooperative software agents find solutions to back-end tracing problems by self-organization. Such cooperative agents are based on a business process aware traceability model, and on a service-oriented composition paradigm. Furthermore, an interface agent assists each user to carry out the front-end tracking activities. Interface agents rely on the context-awareness paradigm to gain self-configurability and self-adaptation of the user interface, and on ubiquitous computing technology, i.e., mobile devices and radio-frequency identification, to perform agile and automatic lot identification. The paper comprises real-world experiences on the fashion supply chain.

1. Introduction

In the last decade, several factors have determined an increasing demand for supply chain transparency [7]. Businesses, consumers and authorities in a variety of markets have today a great interest in establishing systems to trace the history of products in the supply chain. This shared interest has never been more evident than for food, beverage and pharmaceutical products [47], but the key motivations have been expressed in other industry sectors, such as manufacturing and fashion [12,31]. Trading partners in a supply chain can use product traceability to enable different business needs, such as to reduce business risks about legal compliance, to achieve a greater degree of precision and efficiency in product recall or withdrawal, and to comply with specifications of each partner. Further, traceability can be considered also for an efficient logistics and an effective quality management, as well as for supporting product safety, providing information to consumers and partners, verifying the presence or absence of product attributes, such as in the organic food production. Finally, traceability is a fundamental tool for brand protection, product authentication and anti-counterfeit policies [55].

* Corresponding author. Tel.: +39 050 2217455; fax: +39 050 2217600.
E-mail addresses: m.cimino@iet.unipi.it (M.G.C.A. Cimino), f.marcelloni@iet.unipi.it (F. Marcelloni).

Traceability related to products has become to be introduced since the 1990s [9,28] and is still under investigation by scientific and industrial bodies [5,12,20,29]. In [39], a concise characterization of traceability in the batch production industry is provided, together with the notions of chain and internal traceability. More specifically, traceability is defined as the ability to follow a product batch and its history through the whole, or part, of a production chain from raw materials through transport, storage, processing, distribution and sales (called *chain traceability*) or internally in one of the steps of the chain, for example the production step (called *internal traceability*). In [9], traceability in manufacturing systems enables the history of events to be followed and compared with scheduled plans and predefined goals. In particular, tracing techniques can be used in three forms, i.e., to detect system status (*status tracing*), to analyze system performance (*performance tracing*) and to support decision making (*goal tracing*). Each level in which a manufacturing system can be conceptualized, that is, strategy, planning and design, and operation, requires all the three forms of traceability.

In [12,20], the importance of traceability with regard to consumers' perceptions is studied. Traditional determining factors such as price and salary are less important today, and other qualities, called "credence attributes" are becoming relevant to consumers. An example is the increased demand for products labeled with "organic" or "fair trade". Consumers' concern over the safety and ethicality of products has grown in the sequel of contaminations, recalls, and sometimes contradictory research on product health. Supply chains are becoming increasingly complex, since companies generally outsource operations and leverage global sourcing. The key business paradigms to integrate traceability with supply chain management processes have been analyzed in [62]. In particular, traceability has been related to production efficiency, to a better process control, and to the risk assessment models, for identifying various factors that cause quality and safety problems. Furthermore, traceability can play important roles in promotion management and dynamic pricing, with more dynamic and agile planning approaches. Traceability data can also provide instantaneous decision-making responses to variations in the supply chain.

A number of traceability systems, technologies and standards have been developed to carry out chain traceability and internal traceability, with different business objectives [3,6,14,24,31,40,53]. Large enterprises are characterized by a tightly aligned supply chain and typically supported by a considerable use of information and communication technology. Usually, these enterprises already use a traceability system, which typically is very efficient and fully automated [20]. On the contrary, the small enterprises only rarely implement traceability and, when they do, they add the traceability management to their normal operation, decreasing the efficiency and increasing the costs. Indeed, the lack of assets and the difficulty to assess the benefits bring these small enterprises to implement the traceability management in the simplest way, which is often manual or semiautomatic. The characteristics of a traceability system, and mainly its automation level, strongly affect its cost and accuracy [20]. Today, a considerable challenge is to develop agile and automated traceability technology for communities of small-scale enterprises [41].

To fully enable traceability at different manufacturing system levels, for different businesses goals and company sizes, this paper discusses how a chain traceability system with a high level of automation can be developed based on the agent paradigm. Here, cooperative software agents find solutions to back-end tracing problems by self-organization. There are a number of factors which point to the appropriateness of an agent-based approach for supply chain traceability. First, the environment is open, or at least highly dynamic. In such environments, systems capable of flexible autonomous action are often the only solution. Second, trading partners need to work collaboratively in order to achieve the required level of traceability across the entire supply chain, and then agents, known as *cooperative agents*, are a natural metaphor where environments can be modeled as societies of autonomous individuals, cooperating with each other to solve composite problems [23,33,42]. Third, each trading partner in the supply chain has its own objectives in terms of the use of the traceability system, of the level of the smallest traceable item and consequently of data to manage, relative to their trading environment and strategy. This heterogeneity can be faced with an appropriate kind of agents, denoted as *user* (or *interface*) *agents* [65], that simplify the nature of the interaction between user and device, avoiding the configuration of every step that needs to be performed to trace a lot, down to the smallest level of detail, thanks to a certain awareness of the business processes [13]. The idea here is that of considering the agents as assistants of a user in some task [11,34]. The rationale is that current interfaces are in no sense pro-active: activities are only performed when a user initiates a task. Interface agents rely on the context-awareness paradigm so as to gain self-configurability and self-adaptation of the user interface, and on ubiquitous computing technology, i.e., mobile devices and radio-frequency identification, so as to perform agile and automatic lot identification [49]. Fourth, the distribution of data, control and expertise lead to distributed semi-autonomous components [32,65].

The rest of the paper is organized as follows. Section 2 presents a background of the concepts that are relevant to the development of an agent-based infrastructure for traceability. Section 3 details the basic requirements of the traceability system. In Section 4 an architectural view of the agent-based traceability system is provided. Sections 5 and 6 describe the behavioral models of the cooperative and the user agents, respectively. Section 7 is devoted to the application of the system to two real-world supply chains in the fashion sector. Finally, Section 8 draws some conclusion.

2. Agent-based infrastructure for traceability

In an agent-based system, the communication involves high-level messaging in contrast to the low-level messaging typical of a simple distributed computing. The use of high-level messaging leads to a lower communication cost, easy e-implementability and concurrency, and a higher level of autonomy. To this aim, the proposed system relies on a

well-established model of process traceability [3] and on a service-oriented architecture [15]. More specifically, setting up an infrastructure for chain traceability requires working at three different levels. The first level, namely *interconnectivity* level, allows simple data acquisition and transfer with no semantic. This level is covered by technologies such as Radio-Frequency IDentification (RFID) [2], devices such as Personal Digital Assistants (PDA), data format standards such as Electronic Product Code (EPC) Tag Data Specification [14] and data transfer protocols such as HTTP. The second level, namely *integration* level, involves information exchange protocols such as Simple Object Access Protocol (SOAP) [54], messaging technology such as ISO-15000-2 [3] and service composition paradigm such as Service-Oriented Architecture (SOA) [15]. This level enables the agents to exchange information and to request mutually their services. Hence, working at the integration level is not enough, particularly if different enterprises have to compose or expand their business processes. Therefore, the third level, the *interoperability* level, is crucial to a common understanding of semantics about entities and activities. This level represents the top of the traceability infrastructure, and is powered by a model of the business activities of each chain segment, that allow agents to understand and act according to appropriate policies. The integration level reduces the "friction" of interactions between elements belonging to different work cells, whereas the interoperability level enables cooperation allowing the elements to find the solution of a global problem by themselves, coming out as a self-organizing system [21,25]. Hence, agents can represent a powerful tool for traceability, a problem that can be partitioned into a number of smaller and simpler activities, in which each agent is specialized in solving a constituent sub-problem [32]. Structural reuse in different fields is guaranteed by the abstraction level of the process traceability, which addresses a wide range of problems. In the following, we will discuss the interconnectivity and integration levels. The interpretability level will be widely analyzed in Section 3.

2.1. Interconnectivity level

Whenever possible, traceability data should be collected without human intervention. Indeed, humans are typically poor data collectors. Several techniques have been proposed to collect data in a cost-effective manner without interrupting or slowing down the workflow. A widely accepted technique is to associate a tag with each lot: the tag contains the data necessary to identify the lot. These data can be automatically read by appropriate readers.

The most used tags are barcodes and RFID tags. RFID tags consist of a chip that can be attached onto or implanted into any surface of an item [50]. As regards their employment for traceability issues, RFID technology looks very promising [66]. Unlike barcodes technology, e.g., RFID allows acquiring information at a rate of thousand tags per second. Thus, it is reasonable to expect a growing acceptance of RFID technologies in the next years as basic components within traceability information systems [57].

Indeed, RFID technology should significantly reduce both the time and effort needed to execute every-day transactions, in terms of several person-months per year, and the rate of errors that are currently caused by replicated data entries and manual interventions. Retail and manufacturing sectors have considerably benefited from the use of RFID technology in efficiently managing their processes [22,30]. In particular, a number of recent studies indicate that UHF is the dominant frequency used in these sectors [4,10,56]. Further, the integration between a distributed multi-agent system and the RFID technology allows reducing the maintenance effort requested for coping with issues of information asymmetry, decentralized and distributed decision-making, and different data management policies.

On the other hand, RFID is considered a key sensing technology in the vision of "The Internet of Things" [27], promoted by the Auto-ID Labs network [1], together with the Electronic Product Code (EPC) global (EPCglobal) [14], as a paradigm for the supply chain of the future. In this vision, a global application of RFID allows all goods (books, shoes, parts of cars, etc.) to be equipped with tiny identifying devices [19]. Also, a globally distributed information system, made of networked databases and discovery services, allows managing an "Internet of Physical Objects" to automatically identify "any good anywhere". Hence, events like running out of stock or wasted products will be easily detected, since it can be exactly known what is being consumed on the other side of the globe by any customer.

The need to share information in this globally distributed information system requires the adoption of some coding standard which is agreed by all parties and allows them to communicate with each other, so as to ensure the continuity of the traceability throughout the chain. To this aim, the most promising coding system is certainly the GS1 (formerly EAN.UCC) system [24,52], a specification compliant with the EPCglobal Architecture Framework (EPC-AF) [14]. The EPC-AF is a collection of interrelated standards for hardware, software, and data interfaces (EPCglobal Standards), together with core services (EPCglobal Core Services). The EPC-AF neither defines a system architecture which end users must implement nor dictates particular hardware and software components which end users must deploy. EPC-AF only defines interfaces that the end user components have to implement. Thus, EPC-AF makes the end users free to design system architectures according to their own preferences and goals, while defining interface standards ensures that systems deployed by different end users can interoperate and that end users have a wide marketplace of vendor-provided components [14].

Although standardized identification technologies and data carrier middleware are today mature, tracing items in a production chain, across different-scaled enterprises and through the full process scope, is still an inherently expensive design task. Several frameworks have been made available to aid software engineers in storing and retrieving the information flow within a supply chain [5]. The various approaches in the literature are often designed for specific good categories, and are characterized by the need of a top-down design approach for each supply chain. This approach usually produces some specific form of application middleware. Most integration vendors provide methodologies and best practices, but these

instructions tend to be very much geared towards the vendor-provided tool set and often lack treatment of the bigger picture, including underlying guidelines, principles and best practices. In this context, reuse is difficult to attain unless development is undertaken for a close knit range of problems with similar characteristics. General enterprise solutions are more difficult and more costly to develop, because they often need to be tailored to different applications.

2.2. Integration level: from service-oriented composition to multi-agent cooperation

A new phase in enterprise application integration started with the emergence of web services based on XML, a family of interrelated standards which allow program functionalities in different languages and on different platforms to interoperate [3]. Web services are based on a standardized transport protocol (typically HTTP), in which XML structured information are embedded by using standardized messaging protocols, such as SOAP [60]. Web services can communicate with each other in a very robust fashion. Furthermore, each web service is described in a standardized XML schema (WSDL) [61], which can be stored in a repository based on the UDDI standard specification. Web services offer a solution to the challenge of connecting several enterprise applications in a common way, preserving reuse thanks to a mechanism of dynamic search and binding. An innovative paradigm, called SOA, has recently emerged from the concept of web services and other models (such as CORBA [45] DCOM [38], EDI [58]). SOA has rapidly become the blueprint for new forms of automatically composite applications. Composite applications promised a way to use services from a variety of different applications that are exposed through web services. Also, since composite applications reuse parts of other applications, it was possible to separate the process logic from the systems which implement it, thus making possible changes and optimizations of the orchestrated process easier, and helping reduce technology as bottleneck [64]. Further, user interfaces were separated from the application logic, thus allowing composite applications to have user interfaces tailored for different process automation roles. For example, one person might need to see the process from end to end, and another person might need to see one step of the process only. This simple vision of automatically composite applications is particularly popular today for companies selling professional services as well as tools to build web services [26].

The idea of composite applications is not sufficient to reasonably reduce the cost of the chain traceability. Indeed, this idea provides a general-purpose technical integration mechanism. Technical integration refers to the design of technological artifacts that are easy to use as part of larger suite of components, tools and services. Using an enterprise integration approach for traceability, the information systems become more complex, increasing the complexity of the supply chain management. This increase in complexity is characterized by a growth in number and variety of system elements and their interactions [37]. The current method for dealing with this increasing level of complexity is to proportionally increase the number of human system administrators [36]. However, this method is unsustainable at best, as it will eventually make its cost prohibitive.

A more advanced approach involves designing the elements of a system as specialized *software agents* [51] able to find the solution of a specific business problem by themselves, thanks to their awareness of specific business processes. Roughly speaking, an agent provides a convenient and powerful way to describe a complex software entity, which is capable of acting with a certain degree of autonomy in order to accomplish tasks on behalf of its user. Unlike objects, which are defined in terms of methods and attributes, an agent is defined in terms of its behavior [65]. At the integration level, the communication between agents can still be efficiently achieved using the SOA paradigm. At the interoperability level, the agent-orientation principle of autonomy implies that individual agents be as independent and self-contained as possible, with respect to the control they maintain over their underlying business logic. However, to ensure an autonomous processing environment requires a non-negligible design effort. Indeed, some important design issues have to be considered: (i) task decomposition, i.e., how can the traceability tasks be divided and assigned to each agent? (ii) resource management, i.e., how can the agents share context sources? (iii) synchronization, i.e., how can we activate the agents at the right time?. Autonomy is therefore not automatically provided by a web service [46]. To model the tracing agent behavior we therefore have to analyze the supply chain processes so as to determine where and how tracking data has to be collected and therefore to allocate specific tasks to agents.

A fundamental concept in modeling the tracing agent behavior is that of Business Processes. In order to analyze the supply chain processes, these are generally modeled in some specification language, which allows formally describing the activities that need to be performed, the participants who could or should perform them, and the interdependencies that exist between these activities. The activities highlighted in the model permit determining the single tasks for each agent and how these tasks (and consequently the agents) interact with each other. Fig. 1 shows how, starting from a process model (bottom part of the figure), tasks can be allocated to specific agents.

Business logic, explicitly defined for the agent by some set of supply chain processes, is intended to bound the agent task and resource dependencies on other agents or users. Agents use business logic to plan their activities in order to achieve the goal of the concrete process participant. The use of goal-oriented communicating autonomous agents allows multiple solution paths to the process goal to be achieved. Agent-based technologies allow greater flexibility and dynamism in the business process analysis system [8]. Decoupling components of the system allows them to be swapped out, replaced, or even added to the system without impacting other parts. Moreover, there is a decentralized ownership of tasks, information and resources involved in the business process [48].

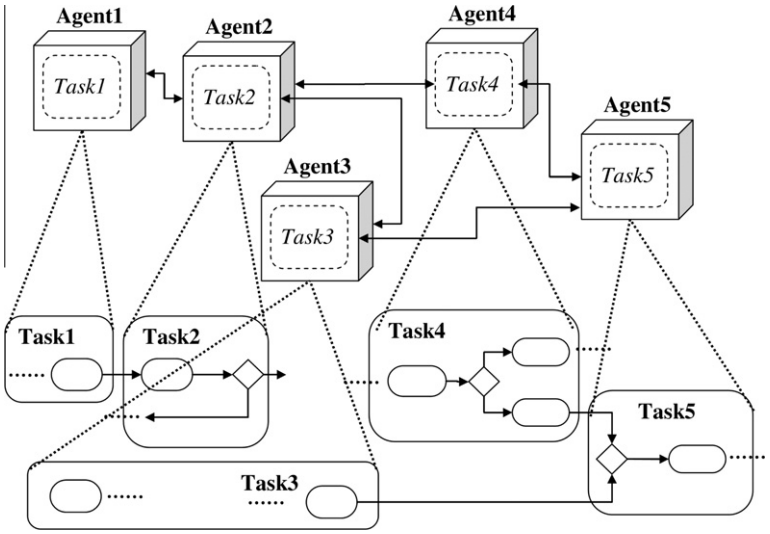


Fig. 1. A topology that employs software agents to process traceability.

3. Agent-based traceability: basic functional requirements

In this Section, first of all we introduce some basic concepts which will be used to define the agent behavior. Then, we address some structural and behavioral aspects of the agents.

In the following, we refer to the data model for traceability we proposed in [3]. Here, two basic entity types have been identified: *lots* and *activities*. A lot is a product unit processed under the same conditions. An activity represents a procedure carried out on some lots (input lots) that produces some output lots. Generic examples of activity are production, packaging or distribution. Assuming that each new lot is generated by an activity, under the control of a person who is responsible for the activity itself and for the corresponding outputs, in the model each lot or activity is associated with a *responsible actor*, actor for short. An *identifier* is associated with each lot, activity, or actor. By means of records of each instance of these entities, it is possible to associate each input lot with the target activity and the corresponding output lots, and vice versa. Hence, activity instances can be chained forward (or backward) by finding their output (or input) lots that are input (or output, respectively) lots of further activities. This makes possible to *trace backward* and *trace forward*, i.e., to follow the downstream path of a product, or to determine its origin and characteristics. In order to have an information flow that is automatically aligned to the physical flow of lots, lot identifiers are carried by means of tags, attached to lots. Fig. 2 shows a UML activity diagram which describes the associations between lot, activity and actor. Here, quality features are also associated with a lot,

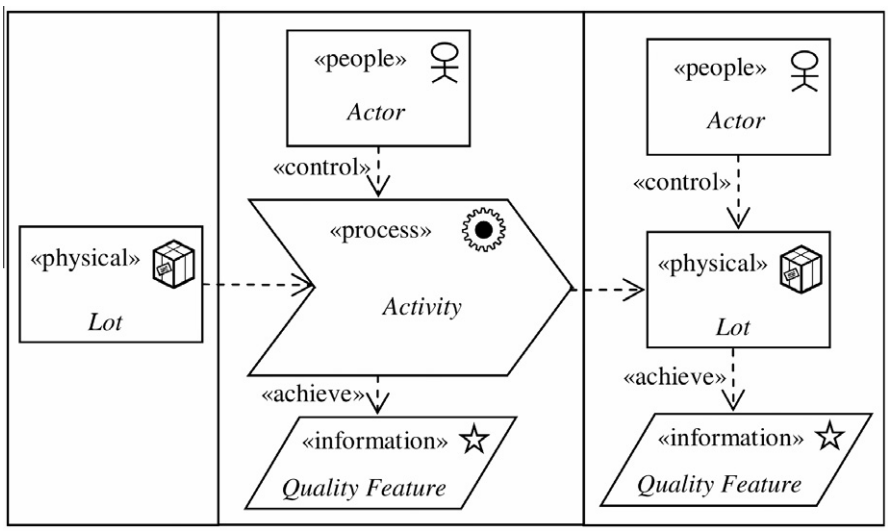


Fig. 2. Activity and its associations with lot, actor and quality feature.

as typical in real-world cases. Examples of such features are weight, rating, temperature, etc. Quality features offer more investigation criteria, from a tracing perspective, with respect to the basic input–output association.

From a tracking perspective, each activity that terminates correctly generates some lots, and for each generated lot a proper business transaction is recorded by the traceability information system. A business transaction is an atomic unit of work that can be associated with the activity. For instance, from an activity with *N* output lots, a set of independent *N* transactions can be tracked. A single transaction cannot be decomposed into lower level independent tracking pieces of information. A business transaction is a very specialized and very constrained semantics designed to achieve product state alignment when needed by third parties. As a transaction, it must succeed or fail, from both a technical and business protocol perspective. If it succeeds from both perspectives, it can be designated as a piece of the lot history. If it fails from any perspective, it should not leave any trace of its existence.

In the following, an exact specification of the content of a transaction is provided. Let us suppose that a *lot* is globally identified by the responsible actor ID (*Axx*), the site ID (*Sxx*), the lot ID (*Lxx*), and the generation date-time (*Dxx*). Similarly, an activity is globally identified by the responsible actor ID, the site ID, and the activity ID (*Txx*). Indeed, considering further constraints, it could be possible to identify a lot with a subset of this data. For instance, let us consider a product with a simple productive process consisting of a number of serial transformations, with no fork and join of activities. If a unique RFID tag is used for each transformation, then the lot ID is enough to identify the lot at each production stage. However, this requirement is very expensive in terms of tags. If a unique tag is used for the entire lot history, then date-time is needed to distinguish the lot at different processing stages. Hence, in each transaction, the lot ID and the date-time are supposed to be necessarily known. The pair (*Lxx*, *Dxx*) allows identifying a lot in a specific stage of the supply chain, even if the RFID tag is re-used after the lot has been sold. To follow the production path, when a new tag is applied to the output lot, it is important to keep track of the input lot ID [3].

Together with the lot, some contextual information is fundamental to support a series of tracing processes, which need to be connected with the real-world at a business level. For instance, when some contamination event occurs, it is important to know *who* and *where* to investigate, and also further features of the lot itself. Hence, in a general traceability model transactions have to contain at least the input/output lots, their site and their responsible actor.

Let us consider in detail how transactions are structured. In the following, the possible transactions are discussed.

3.1. Providing-acquisition

Fig. 3 represents a scenario of providing-acquisition of a *lot*. At the instant *D1*, the actor *A1* provides the actor *A2* with the lot *L0*, which was stored at the site *S0*. At that moment, *A1* could not know the site in which *A2* will store the lot, and then, in her/his vision, that site is denoted by *S?* (unknown site). This is usual, for instance, if the two actors belong to different companies, or if some module has not been properly configured. In this case, the transaction will have an undefined output site (transaction *TR1* in Fig. 3).

Similarly, at the instant *D2*, the actor *A2* acquires the lot *L0* and stores it in its own site *S3*. However, he cannot know where the lot was previously stored. Again, in this case the acquisition transaction will have an undefined input site (transaction *TR2* in Fig. 3). Note that, in Fig. 3, the lot is identified by two different RFID tags before and after the acquisition, i.e., *L0* and *L1*, respectively. On the other hand, if the RFID tag is kept, *L0* will be equal to *L1*. Note how, starting from the input lot of the transaction *TR2* (i.e., [*A1*, *S?*, *L0*]), and replacing its actor (i.e., *A1*) with the actor in the output lot (i.e., *A2*), it is possible to derive the output lot of the transaction *TR1* (i.e., [*A2*, *S?*, *L0*]). This means to identify the transaction *TR1* with some data available in the transaction *TR2*, i.e. a step backward in the tracing. If more than a transaction with the same output lot is available, the transaction *TR1* closest in time to *TR2* is considered (i.e., with *D1* such that *D1* is closest to *D2*). Vice versa is also valid for a step forward (tracing forward).

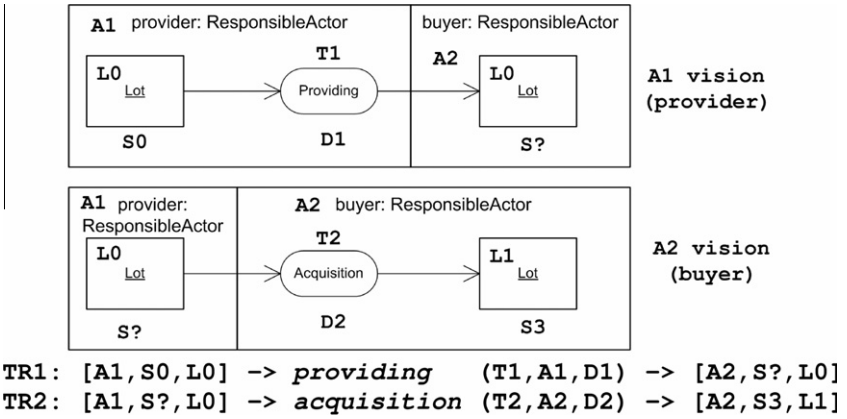


Fig. 3. A scenario of the *providing-acquisition* transactions.

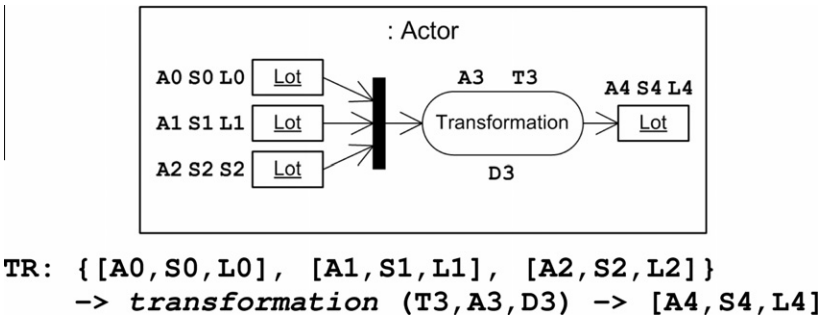


Fig. 4. A scenario of the *transformation* transaction.

3.2. Transformation

In the case of processing activities that are internal to a company, a group of *N* lots can be transformed into a group of *M* lots, via splitting, merging, moving, processing, etc. This activity can be represented as a series of *M* transformations of *N* lots into a lot, having the same lots as input. Fig. 4 describes a scenario with three input lots. Here, at the instant *D3*, the actor *A3* performs the activity *T3*, taking as inputs the three lots, *L0*, *L1* and *L2*, and giving as output the lot *L4*. The input lots were stored at the sites *S0*, *S1* and *S2*, respectively, and owned by the actors *A0*, *A1* and *A2*, respectively. The output lot is stored at the site *S4*, and owned by the actor *A4*. Note that, in this transaction, tracing backward and forward are simpler to perform with respect to the providing-acquisition transaction, because sites are known.

4. Agent-based traceability: architectural view

In this Section we consider more specifically the architectural view of the traceability system. Fig. 5 shows a deployment diagram containing different kinds of units: some of these units host software agents. The proposed traceability system comprises different *Tracking Units* (*TUs*) equipped with RFID readers. A *TU* gathers data and transmits them to a *Storing Unit* (*SU*). *SUs* are in charge of keeping local production data, supplied by *TUs*, according to some criteria. *Analysis Units* (*AUs*) steer business process analyses and harvest data supplied by *SUs* in terms of pieces of a global tracing problem. *TUs* can be hosted by a mobile device (e.g., PDA or smart phone equipped with an RFID reader), or fixed device (e.g., bank reader, door gate reader). Further, *TUs* allow data harvesting supported by user agents, because *TUs* are self-configured on the basis of the local context. More specifically, there are some *Context Units* (*CUs*), which are able to provide a local business process context. Indeed, *CUs* and *AUs* are strictly related to each other. For a given business analysis, a set of data needs to be collected, and this can be defined configuring *TUs* via *CUs*. Furthermore, *CUs* contain also the definitions of the quality features used by *AUs*. Finally, there are some lookup services for *SU*, accomplished by *Registry Units* (*RUs*). Only *SUs* and *TUs* host agents. In particular, an *SU* hosts a cooperative agent, and a *TU* hosts an interface agent. We will explain these agents in Sections 5 and 6.

As shown in Fig. 5, the proposed traceability system is based on a distributed architecture. Here, data is managed according to a "pull" model [3]. In the pull model, at the *tracking stage*, data is stored at the site where it was generated, via *TUs*. An RFID reader is associated with each *TU*, and controlled by the user agent. At the *tracing stage*, an *AU* actively requests a particular analysis from the system. Hence, *SUs* wait for a pull request to reconstruct a lot history. When a pull request arrives, only related tracing data is collected and returned to the *AU*. As summarized in Fig. 5 by the cardinality ratios, the number of *TUs* is typically much larger than the number of *SUs*.

An overall dynamic view of the system is represented by the communication diagram of Fig. 6. Here, the most important relationships between the system units are summarized. In particular, a unit which is able of concurrent control of the communication is represented with doubled vertical sides.

Let us consider a typical scenario. Let TU_1 be a specific *TU* and CU_1 the corresponding *CU*. Once powered on, TU_1 sends its ID to CU_1 which returns the appropriate context to TU_1. The context includes quality feature definitions, responsible actors, activities and sites related to TU_1. This allows the self-configuration of TU_1. The production context is also important as implicit input, to reduce the manual data input and the multiple choices available to the user in the *tracking* phase. Furthermore, the production context allows the self-adaptation of the user interface. This mechanism will be described in Section 6.

Each *CU* is aware of a little segment of the whole production, and hence it handles a local model of the activities. Once TU_1 is configured, in the *tracking* phase it can harvest a number of data regarding local production. This amount of production data, as described in Section 3, is built in terms of transactions and pushed by TU_1 to the storing unit SU_1, which TU_1 sends the data to. During the *tracking* phase, no collaboration is established between *SUs*. SU_1 records a series of tracks, coming from the related *TUs*, for instance TU_1, according to predefined business objectives. Such business objectives are strictly connected to the business analysis that a specific analysis unit AU_1 can perform. Hence, for a given business analysis, a set of tracking points is established. On the contrary, the tracing phase requires collaboration among *SUs*.

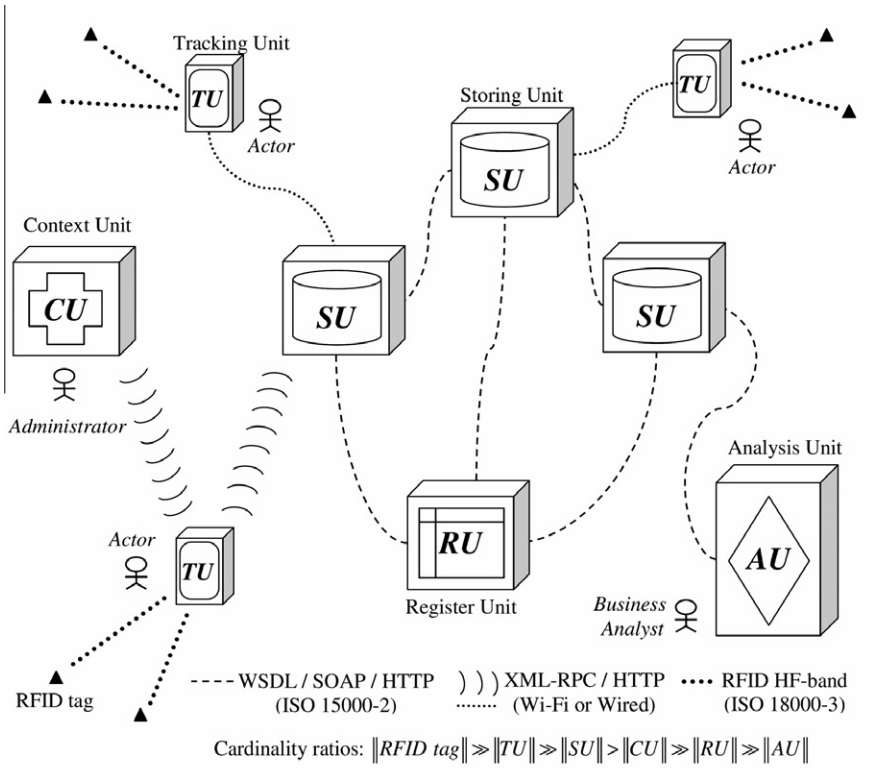


Fig. 5. An overall deployment diagram of the traceability system.

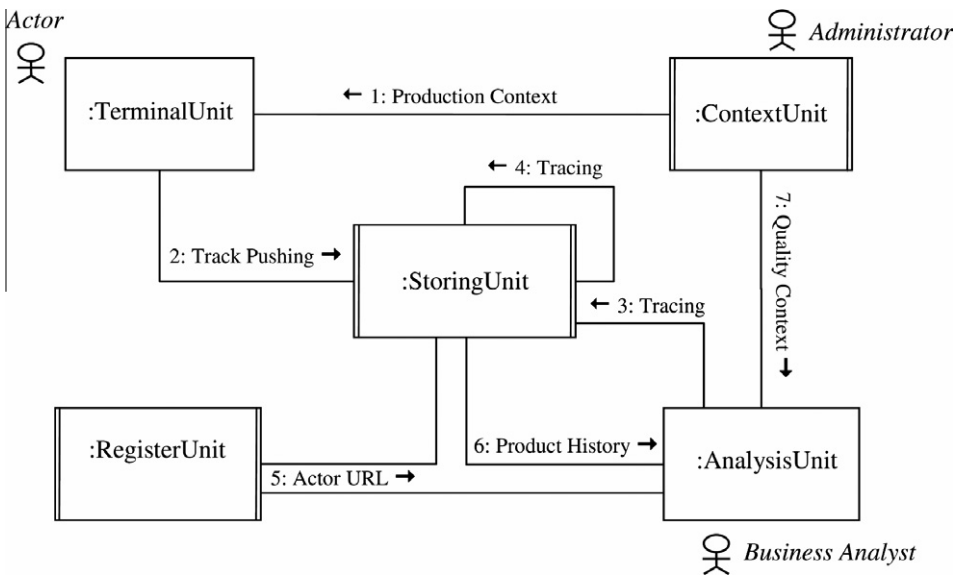


Fig. 6. A communication diagram of the traceability system.

Let us suppose, in the scenario, that AU_1 asks for the overall history of a specific lot. AU_1 is supposed to know the lot identifier, e.g. *L1*, the responsible actor identifier, e.g. *A1*, of that product, and a register unit RU_1, a front-end register. Hence, using an SOA-based dynamic binding paradigm [3], AU_1 sends the identifier *A1* to RU_1 and receives the URL of SU_1 from RU_1, where the tracks of *A1* are stored. Then, AU_1 asks SU_1 for the history of *L1*. SU_1 performs a search in its repository, and sends all

available data to AU_1. This is just a shallow history, i.e., concerning the activities performed by an actor in the last supply chain segment. In the case of deep history, SU_1 is able to trace backward and follow the production path, finding other responsible actors. For instance, let us suppose that the responsible actor *A2* accomplished the penultimate activity on the product. If *A2* is locally stored in SU_1, its search is very similar to the search of *A1*. Let us suppose that *L2* is remote, i.e., stored, for instance, in SU_2. Then, SU_1 asks to RU_1 (let us suppose that the register is shared) about the URL of the *SU* where *L2* is hold. RU_1 provides this URL and then SU_1 sends a request to SU_2. It is worth noting that SU_2 is not aware of SU_1, because the request is done "on behalf of" AU_1, which appears as sender. Hence, SU_2 will send tracks directly to AU_1. In a recursive fashion, it is then possible to trace backward and trace forward in the production chain, as shown in Fig. 6. Please note that many possible relationships could be established in this recursive search, as described in Section 5. For example, a deep search could be performed considering only defective products.

According to the SOA paradigm, the communication between *SUs* and *AUs* relies on an asynchronous message-centric protocol, which provides a robust interaction mechanism among peers, based on the SOAP/HTTP stack. On the other hand, the communication between *TUs* and the other units can be proficiently achieved using a more efficient and lightweight XML-RPC/HTTP based interaction. The implementation of the SOA paradigm in the proposed system is based on the ISO 15000 standards, which comprise a Messaging Service Specification (MSS), a Registry Information Model and a Registry Service Specification (ISO 15000-2, 15000-3 and 15000-4, respectively). Each agent of the traceability system is supplied with an MTS based on MSS to contact a registry and/or another agent. In the framework, an agent which desires to send a message to another agent or to request a service uses the Directory Service (a federated registry [16]) to obtain a set of descriptors for available agents and services. Once these descriptors have been retrieved, the agent communicates by exchanging messages encoded in the specific Agent Communication Language (ACL) used in our system and delivered by the appropriately defined Message Transport Service (MTS) [18]. The directory and the MTS are completely independent of the behavior embodied in agents. This behavior is characterized by some common features [35]: (i) *social ability*: tracing agents are able to interact with each other via an ACL; (ii) *reactivity*: agents are able to perceive their environment, through *TUs*, *SUs* and Internet, and to respond in a timely fashion to changes that occur in it; (iii) *pro-activeness*: a tracing agent not simply acts in response to the environment, but it is also able to exhibit goal-directed behavior by taking the initiative, according to a collaboration protocol; (iv) *autonomy*: a tracing agent operates without the direct intervention of humans or others, and controls its actions and internal state.

As regards the latter feature, it is worth explaining the rationale of the intrinsic autonomy of each agent. The agent behavior is locally established and configured by the company which owns data retained by the agent, following specific policies. Indeed, supply chain participants, at all segments of production, are often highly protective towards their own data, thus they would not agree on sharing all their data on the same agent with the others. Ownership, movement and location data might be used for purposes different from traceability. Thus, each agent acts on behalf of a local management, in order to avoid issues of data confidentiality, trade disruption and data integrity. This degree of autonomy is carried out by a cooperative pattern between agents, in which each agent assesses the utility of a collaboration before accepting it.

In our traceability system, the collaboration between two agents is ruled by the ISO 15000-1 standard, which allows specifying an agreement between partners. Each partner has its own collaboration profile, which describes a series of requirements in an XML format. For instance, this can include the role of the partner: if the partner does not match the profile determined by the agent, such agent can refuse the collaboration. The agreement can also specify the quality of services guaranteed by the partner: supported messaging protocols, security capabilities, rules to follow when acknowledgments are not received (including how long to wait for before resending, and how many times to resend), whether duplicate messages should be ignored, whether acknowledgments are required for all messages. Of course, a collaboration profile has to take business failure conditions into account too. Transport level failures are managed by the MTS, which caters for reliable and recoverable message exchange. The collaboration profile handles the business level failures. For example, if a partner fails to reply within a pre-defined time interval, then the agent reverts to the previous known secure state of the collaboration. The message-exchange agreement between two partners is described by a collaboration protocol, which can be defined as a series of communicative acts. In the following, the basic communicative acts between tracing agents are discussed.

In the collaborative protocol, an agent (the Initiator) takes the role of manager which wishes to have some traceability task performed by one or more other agents (the Participants). The representation of the protocol is given in Fig. 7 by a UML 2 sequence diagram. The protocol can be considered as an adaptation of the FIPA pattern to a simple interaction type [17].

The Initiator solicits a proposal from an agent by issuing a call for proposal (*cfp*) act, which specifies the task and the conditions placed upon the execution of it. The Participant agent generates a response, which can be a *propose* act or a *refuse* act. This decision is based on a utility function which takes both the proposal requirements and the specific agent policy into account. The *propose* act contains a proposal to perform the task. This proposal includes the preconditions that the Participant is setting out for the task, which may be the quality of service, time when the task will be done, etc. The *refuse* act communicates to the Initiator that Participant will not perform the task. The Initiator evaluates the proposal of the Participant, if any, and sends it an *accept-proposal* act or a *reject-proposal* act. After the Initiator accepts the proposal, the Participant acquires a commitment to perform the task. Once the Participant has completed the task, it sends a final message, which includes local tracing results and, if required, a new call for proposal to another agent, started on behalf of the Initiator. This agent replies directly to the Initiator by using the same propose or refuse acts, thus using the same collaboration protocol described above.

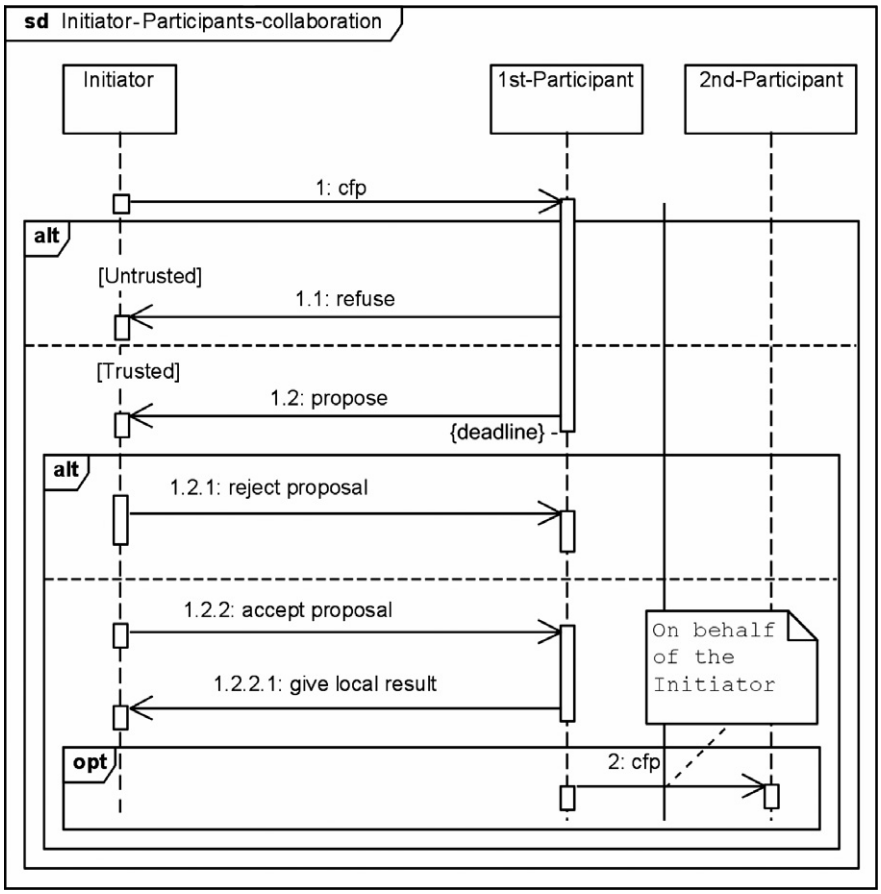


Fig. 7. The collaboration protocol between tracing agents.

The ACL of the system specifies which parameters are needed for effective agent communication. A concrete instantiation of an ACL is a mandatory element of any multi-agent architecture [18]. The basic parameters used in the system are: *performative* (i.e., type of communicative acts), *sender*, *receiver*, *reply-to*, *content* (i.e., content of the message), *protocol* and *conversation-id* (used for controlling the conversation).

In tracing agents, the main task to perform is a search for items with proper features and conditions. As a matter of fact, any service requested by an *AU* can be decomposed into a series of dynamically coordinated local searches. Hence, the task is represented by using a syntax for expressing queries. The syntax used in the framework is based on XSLT with XPATH 2.0 (XML Path Language, W3C recommendation 23 January 2007). The main advantage of this syntax is the possibility of automatic translation into SQL syntax, which can be executed by a conventional relational DBMS. Hence, the implementation of different searches and constraints can be easily managed with a general-purpose and efficient parser and a relational engine.

A search for items employs a series of different lot features, qualitative and quantitative, which are constrained in terms of possible values. For instance, employing a multi-valued or interval-valued (i.e., set of possible) search key, thanks to the use of regular expressions. In general, the search expression comprises mandatory attributes and alternative attributes as well as optional attributes.

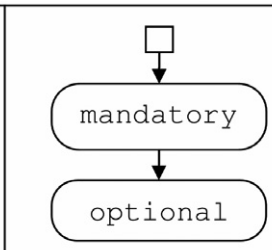
Fig. 8 shows an example of how the parser builds an internal data abstraction, for two basic patterns, useful to be translated into SQL expressions. In particular, the patterns show how mandatory and optional sets of attributes are connected by a father-son relationship, whereas optional elements of a set are connected by a sibling relationship.

Other important features of conditional expressions built with XPATH2.0 are:

(i) the use of regular expressions, with the following form:*fn:matches (*$*input as xs:string,* $*pattern as xs:string)*;
(ii) the use of date-time conditions, with the following form:$*generationDate*& *gt;=yyyy/mm/dd hh:mm:ss" AND*$*generationDate*& *lt;=yyyy/mm/dd hh:mm:ss") OR*$*startingDate*& *lt;="yyyy/mm/dd hh:mm:ss)*;
(iii) the use of conditional expressions with quality features, expressed in terms of name and value;
(iv) the use of conditional expressions with variables, using the < xsl:variable > element.

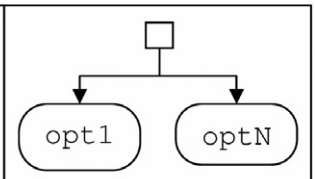
Fig. 9 shows a simplified example of query translation.

```
<root>
 <xsl:if cond="(mandatory attributes)">
  <xsl:if cond="(optional attributes)">
   <result/>
  </xsl:if>
 </xsl:if>
</root>
```



(a)

```
<root>
 <xsl:if cond="(opt1... OR ... optN)">
  <result/>
 </xsl:if>
</root>
```



(b)

Fig. 8. An example of how the parser builds structured query conditions.

```
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
   <xsl:for-each select="/messageData/traceableEntities/lot">
     <!-- variables declaration -->
     <xsl:variable name = "id" select = "./id" />
     ...
     <!-- mandatory attributes -->
     <xsl:if cond="( (matches($description,'FAULT*') AND ... >
     ...
     <!-- optional attributes -->
     <xsl:if cond="(($responsibleActorId='TAYLOR' OR ...>
      <result>
       <xsl:copy-of select="."/>
      </result>
     </xsl:if>
     ...
     </xsl:if>
   </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

(a)

```
SELECT *
FROM 'tracing'.'lot'
  LEFT OUTER JOIN 'tracing'.'qualityfeature' ON
                  lot.lotid = qualityfeature.lotid
WHERE (qualityFeature.description LIKE 'FAULT*' AND ...
...AND (((lot.responsibleActorId='TAYLOR' OR ...)
ORDER BY lot.lotid
```

(b)

Fig. 9. An example of query translation from XSLT to SQL.

5. Cooperative agents for tracing: a behavioral model

The behavioral model of the tracing agent *SU* is based on the basic transactions described in Section 3. Transactions are managed in the tracking phase, and the corresponding data are sent from the *TUs* to the *SUs*, in the form of XML-based

documents. In the following, more detailed specifications, regarding *SUs* and the tracing phase, are provided. A set of interesting features have been designed and implemented for *SUs*, in order to satisfy many business needs. As an agent, an *SU* controls also some internal maintenance features, such as garbage collection, data migration, register caching, and so on. In the following, we list some of these features.

5.1. Attribute-driven search

It is possible to ask the agent for lots belonging to a given responsible actor, to a given site, or to a combination of site and responsible actor, i.e. the basic lot attributes in the model. Further, it is possible to provide more than a possible search value for each attribute, using regular expressions. An important lot attribute is called *product-code*, i.e., a mnemonic code used by a firm to identify a product. Usually, it is possible to use the product-code as a foreign key to gain a lot of class-related information about the lot, stored on information systems different from traceability systems. Another common search is based on the date-time, i.e., to filter the search for starting or ending date within some time interval.

5.2. Direction: backward/forward

In a distributed system, it is clear that the agents should be able to collaborate with each other so as to explore the lot history by tracing backward and forward in the supply chain. It is simple to demonstrate that each production path creates a distributed relationship among different *SUs*. This relationship can be represented as a directed acyclic graph, where the nodes and the arcs represent the *SUs* and the communication paths among the *SUs*, respectively. The relationship is a *graph* because of the splitting and merging activities between lots. The graph is *directed* because of the input–output relationship of each activity. In order to show that the graph is *acyclic*, let us consider Fig. 10.a. Here, a simple production flow scenario is represented, considering the physical (on the left) and information (on the right) flows. The execution of each activity is identified by a counter, in both flows.

In particular, the same activity, represented by a white gear on the left, is carried out twice (execution n. 1 and n. 2). However, each execution produces a different information lot on the right. It becomes clear that, even if the same *physical* lot goes though cyclic activities, each execution of an activity generates an output *information* lot which is different from the input *information* lot. Furthermore, considering that any input physical lot is generated before the correspondent output physical lot, the generation time of an input information lot precedes the generation time of the corresponding output information lot. It follows that cycles are not possible in the information graph.

This property is important to avoid the emergence of unexpected global behavior of the society of cooperative agents, one of the drawbacks of the distribution of control in collaboration paradigms. Let us consider Fig. 10.b, representing an example of production flow among *SUs*. Let us suppose that an *AU* asks an *SU* for a complete lot history, starting from a specific lot. If the direction is "backward" or "forward", there is no possibility of cyclic interaction between agents, as the graph is acyclic. Let us suppose that SU_4 is asked for a complete history, in both directions. Hence, starting from SU_4, SU_{17} and SU_8 are contacted, and then SU_6. But, starting from SU_6, SU_8 is contacted again, and so on, with an infinite loop. This situation is called *livelock*. The states of the agents involved in the livelock constantly change with regard to one another, none progressing. To avoid the livelock, each agent stores a *business signature* of the message for a reasonably limited amount of time. The signature comprises the sender ID, which is an *AU*, and the message ID, which is provided by the messaging system. Hence, if an *SU* receives a second message with the same signature, the message is discarded.

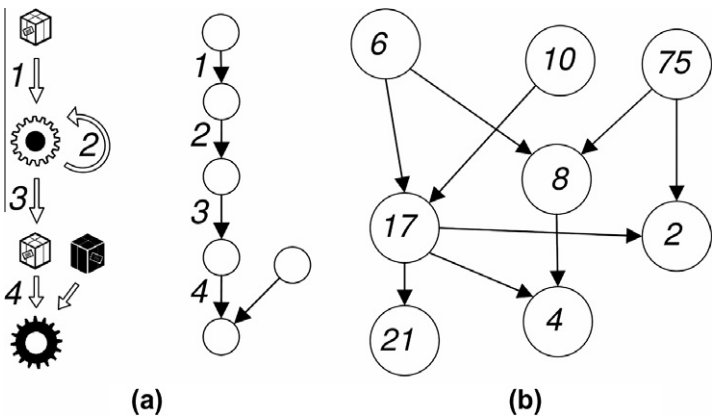


Fig. 10. (a) Distinction between physical flow and information flow; (b) an example of production flow among storing units.

5.3. Shallow/deep search

It is possible to ask the agent to stop the search after a predefined number of activities starting from a certain lot. This can be done using an *activity counter*, which is transmitted and decreased by each agent. Once it is zero, the agent will stop the search.

5.4. Copy/move mode

In the *move* mode, the agent removes its own data once it has been provided to another agent or to an *AU*. In the copy mode, data is kept, providing the requester with a copy. The move mode is important to implement data migration strategies.

The amount of accumulated data at each *SU* can be controlled through two different services: garbage collection and data migration. An internal garbage collector operates locally by periodically searching for inconsistent or expired transactions. Further, a special *AU* is responsible for removing expired or unnecessary transactions, considering specific policies: for instance, all the transactions can be removed after a fixed time interval or once a backup process has been performed. Finally, another *AU* controls the data migration, following other specific policies. For instance, the *AU* can decide to transfer data from an *SU* to a local repository for improving the efficiency of specific analyses.

6. Interface agents for tracking: a behavioral model

The objective of the interface (or user) agent is to help users in various kinds of situations, which change dynamically, employing different preferred features according to the particular context. The input–output functionalities are limited on mobile handheld devices, because of the small size of the device. Hence, process-awareness aims to use the information of the usage context for adapting the behavior of the application according to an appropriate model, and to facilitate the success of the business transaction.

More specifically, the traceability context of the agent behavioral model comprises the parameters summarized in Fig. 11.

In designing the user interface, several users involved in the supply chain were interviewed. From these interviews, two important considerations were derived. First, *TUs* are usually managed by workers with limited experience in ICT devices. Second, the use of *TUs* should not hamper the normal current activity. Hence, a *TU* should be able to offer an interface with the lowest level of complexity in dependence of the context. For example, if there are only two possible choices in a specific production segment, a related menu should offer only those choices, considering as default the most frequent choice. If some specific step or parameter is not necessary, it should completely disappear.

To achieve this objective, the use of a touch screen seems to be more appropriate than a keyboard and/or a mouse. Further, the use of keyboard-based inputs and of any intermediate device, such as a stylus, that needs to be held in the hand should be avoided. Finally, the layout preferred by the mass of the users is a *grid* layout with a top status area, i.e., similar to the layout of a common pocket calculator. This choice is optimal for recurrent use, in which the same transaction is accomplished for a large number of times, as usual in workflows. Once established a layout, an important requirement to address is the dynamic placement of components during a transaction. The use of a touch screen allows a wide self-adaptation of the graphic components, presenting at each time only what is needed. However, it has been experienced that the use of changeable buttons during a transaction is very confusing for the users. Hence, although several possible transactions, which require different buttons, are possible, in each transaction all needed buttons should be fixed in advance, to keep a good awareness of the status. This allows skilled users to gain an excellent automatism in a short time. On the other hand, to guide unskilled users, the agent can disable the useless buttons, keeping them still visible. Fig. 12 represents the layout of three different interfaces for three different situations. Each interface is related to a different transactional situation. Obviously, when some choices can be automatically determined by the agent, the interface shown to the user is

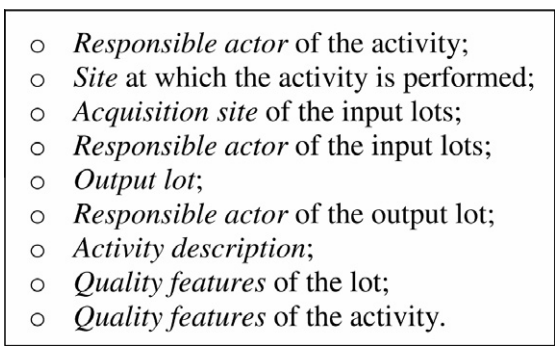


Fig. 11. Traceability context for an interface agent.

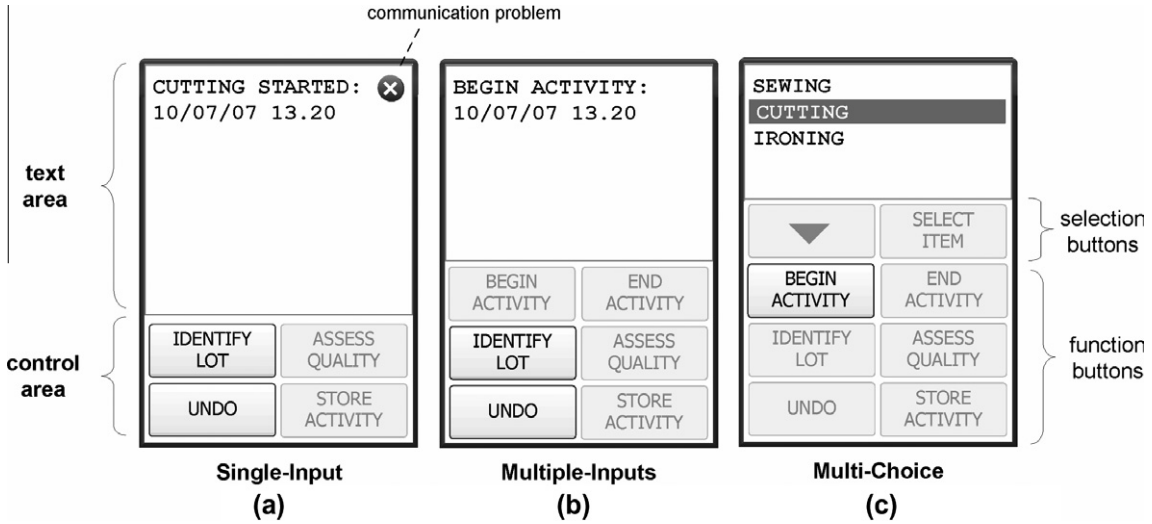


Fig. 12. Context-aware interfaces of a *TU* for different situations.

simplified. Furthermore, the agent is also able to customize the buttons labels, providing more detailed labels for each situation. In Fig. 12, generic default labels are represented.

The lowest complexity is represented by the *transit* situation, which considers just the transit of a lot: this situation requires no explicit input parameter at all. The adoption of the transit situation is usual for bank or gate reader. In this case, the agent derives everything from the traceability context: the tag ID, the date-time, the actor, the site. A higher level of complexity is represented by the *single-input context* shown in Fig. 12.a, where the activity is supposed to have a single input, a duration and a binary-valued quality factor (e.g., good/faulty). Under these circumstances, the user usually presses only the *identify lot* button to read the RFID tag (and then to implicitly begin an activity) and the *store activity* button to commit the transaction at the end of the activity. The *assess quality* button, in this situation, allows recording a faulty output lot, once pressed.

Fig. 12.b shows a possible layout for a *Multiple-Inputs* context. The activity starts and terminates when the user selects the *begin activity* and *end activity* buttons, respectively. During the activity different input lots can be identified. The *assess quality* button acts as in the *single-input* context.

Fig. 12.c shows a possible layout for the *Multi-Choice* context. This means that there are attributes with multiple choices, e.g., many activities (such as sewing, cutting and ironing) can be performed by an actor, and that many inputs for the activity are possible. Hence, when the activity needs to be chosen, the agent is provided by the traceability context with the possible activities only for that actor and for that type of lot. Moreover, the activities list is presented in frequency order. Furthermore, the *assess quality* button, once pressed, allows the selection of specific values related to the context. In other terms, the ultimate goal of the agent is to avoid a manual input or selection, using all the available information.

Fig. 13 shows a simplified State Diagram of a *TU* for the *Multiple-Inputs* and *Multi-Choice* situation, which represents the behavior of the *TU* application controller. Many State Diagrams are possible, depending on the traceability context and the traceability situation. Hence, self-adaptation and self-configurability are the most important properties of this agent. On the basis of the specific tasks the *TU* has to manage, the *CU* sends the appropriate context to enable a tailored application flow on the *TU*.

Once completed, a transaction cannot be rolled back by the *TU*. It is locally buffered and then sent to the *SU*. Other contextual behaviors are managed by the agent. For instance, it provides an adaptive communication with the SU. The sending rate is adaptive, depending on the availability of wireless connection, to optimize the energy consumption. Transactions are temporarily kept on persistent memory by the agent, and then sent once connection is again available. On the user interface, a special alert symbol (on the top right of the single-input interface in Fig. 12) is presented to the user to highlight communication problems.

7. Experimental studies

The presented architectural model, namely Mobile and Agent-based Traceability (MOAT), has been developed and experienced under research and innovation programs for the development of the Tuscany small-medium enterprises. The MOAT system has been applied to real-world firms on the fashion supply chain. In this Section, more technical details are provided, together with some applicative scenario, in order to show how traceability allows gaining concrete insights about chain processes and to enhance decision making activities [59].

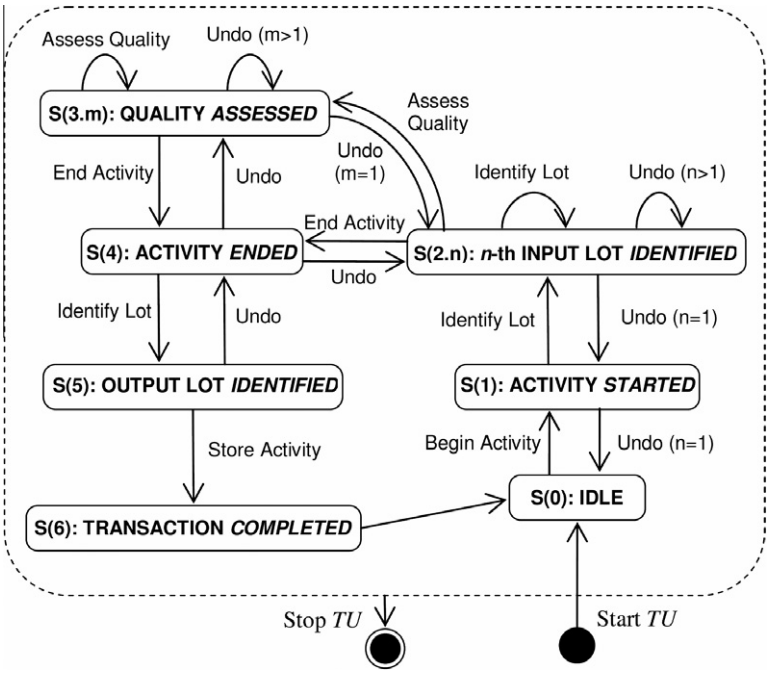


Fig. 13. Simplified state diagram of a TU for the multi-choice context.

The MOAT system has been developed with public domain software, or software natively provided with the hardware devices, in order to be completely costless in terms of licenses for the factories joined to the research program. In Tables 1 and 2 the most important software and hardware products are detailed.

To describe the specific working context, Fig. 14 shows a Business Process Modeling Notation (BPMN) [44,63] representation of the process for manufacturing a bag in a real leather chain. Fig. 14.a outlines the macro processes, from the bag design to its shipment. Fig. 14.b represents a drill down through the "check and ship out" sub-process, where semi-finished products originated from the "assemble" sub-process are checked against quality. If the products are good, they are packed and shipped out; otherwise proper corrective actions are triggered to handle the error. Finally, Fig. 14.c shows the expansion of the "check product" sub-process: in the first gateway, features of the product are compared with specifications and quality plan. If the product is compliant, it can be packed and shipped out. Otherwise, the product is analyzed so as to determine the

Table 1
Software products used for the MOAT system implementation.

System component	Software product	References
Storing unit,	Java SE 1.6	http://dev.mysql.com
Context unit,	MySql 5	http://java.sun.com/javase
Register unit	Apache Tomcat 6	http://tomcat.apache.org
ISO 15000-2 interface	Hermes Message Service Handler 2	http://www.freebxml.org
Terminal unit	Microsoft Windows Mobile 5	http://www.microsoft.com
	.NET Compact Framework 2	http://www.microsoft.com
Analysis unit	Compiere ERP 2	http://www.compiere.com
	Oracle 10g	http://www.oracle.com

Table 2
Hardware products used for the MOAT system implementation.

System component	Hardware product	References
Terminal unit	HP Ipaq HX2490 PDA	http://www.hp.com
(for indoor use)	Compact Flash Reader: CFR-F13-56	http://www.rf-id.it
Terminal unit	Psion TekLogix, with OEM	http://www.psion.com
(for outdoor use)	COM3 HF Reader: CPRM02	http://www.rf-id.it
RFID HF tag	Work-Tag Film labels	http://www.rf-id.it
Wireless gateway	Linksys WAG300N	http://www.linksysbycisco.com

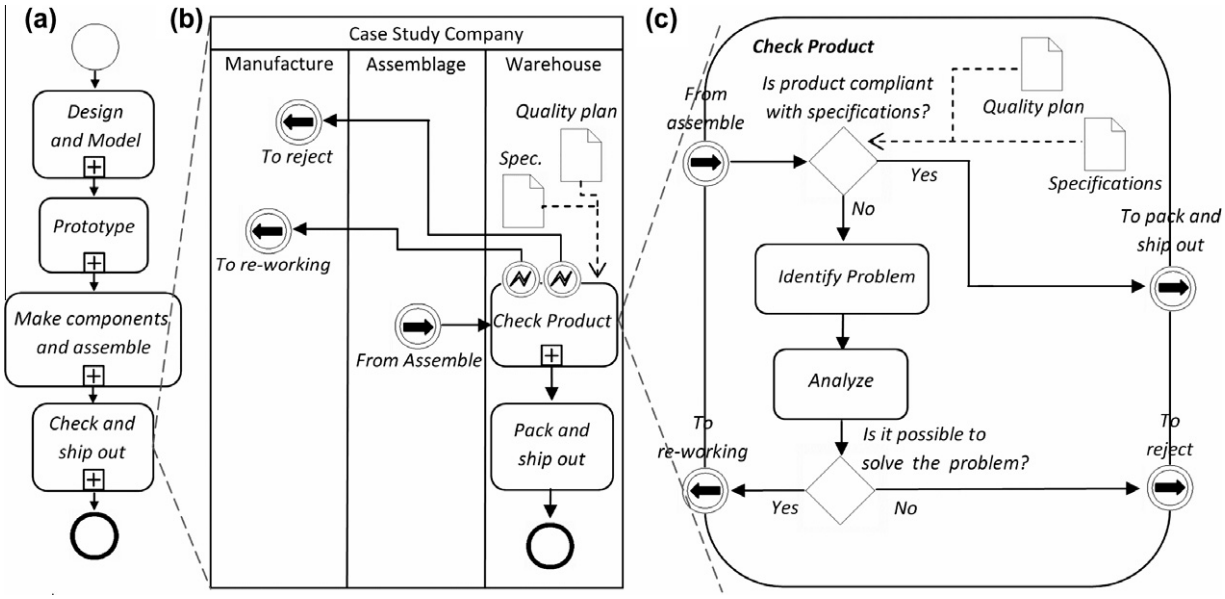


Fig. 14. A BPMN business process flow representation, for the production of a bag.

causes of the fail and to examine the possibility to remove the defect. If the defect can be removed, then the "re-working" sub-process is fired, otherwise the product is dispatched to the "reject" sub-process.

In the following, two examples of business analysis enabled by traceability are detailed, namely the improvement of outsourcer selection and the analysis of the value stream in the supply chain.

As regards the outsourcer selection, the system has been used to trace the most important parameters to support this decision. Typically, the outsourcers are used to manufacture the bag after cutting the leather. MOAT allows determining the lead time spent for each activity in the bag production. In particular, for a set of crucial activities, we collected information about the average lead time, variance of lead time, the average price and variance of price. MOAT allows business analysts to know the lead time spent for certain phases of the manufacturing process. Table 3 shows gathered information concerning the production of a high quality bag. The considered activities sweep from the processing of the leather, through the cut of the bag elements, to the assembly and packing of the end product. Depending on the chosen outsourcer, duration times and costs can change accordingly. Furthermore, other quality parameters have been modeled to better cope with the outsourcer selection problem. In particular percentage of defects and average delay in signaling defects have been traced in order to give executive managers the chance to select the suitable outsourcer.

Furthermore, MOAT allows evaluating the lead times of the value-added and non-value-added activities. Thus, business analysts can determine possible causes of waste and reduce total lead time within the supply chain. To this aim, all activities of our case study have been pre-grouped into *manufacturing*, *warehousing*, *waiting*, *transport* and *inspection* processes. For each process, we traced average lead times as shown in Table 4.

This information has allowed management to investigate causes of waste (e.g., waste of time for waiting, waste of space for warehousing, waste of workforce for transport) and to identify improvement areas [43].

A traceability approach which is service-oriented, but does not employ agents, has been discussed in [3]. Here, decentralization concerns solely data, whereas the control over data is centralized, and owned by the *AU*. On the other hand, if all *SUs* are services and therefore are not autonomous, only a global shared policy is possible in the supply chain. Actually, in terms of data harvesting, the process of tracing would not require agents, but the lack of local autonomy would not offer to the owner alternatives out of the common behavior for data sharing. On the contrary, in the MOAT system, the agent-based paradigm allows an autonomous control over local data. Thus, the MOAT system can be considered an evolution of the system discussed in [3]. In particular, we have replaced this system with the MOAT system in the supply chain.

Table 3
Data collected from process tracing services concerning the production of a high quality bag.

Activity	Average lead time (min)	Δ Lead time (min)	Average price per minute (Euros)	Δ Price (Euros)
Cutting	25	±3	0.32	±0.05
Predisposition	28	±3	0.25	±0.05
Assembly	135	±10	0.29	±0.03
Packing	11	±2	0.28	±0.02

Table 4
Average lead time for each process.

Activity	Time (min)	Percentage
Manufacturing	2880	53.51
Warehousing	1920	35.67
Waiting	480	8.92
Transport	95	1.77
Inspection	7	0.13
Total	5382	100.00

The most immediate benefits of the MOAT system have been a sensibly reduced training cost and an increased supply chain stability, thanks to decision making personalization and configuration automation features. In particular, the auto-configuration process performed by the *TUs* via the *CUs* on the basis of a declarative description of the productive processes has allowed considerably reducing the effort needed for setting-up and maintaining the system. Further, the use of user agents with the aim of automatically adapting the *TU* user interfaces has sensibly improved the usability of the system. Thanks to the agent autonomy, companies have improved the management of their local control through the *SUs* that are able to adopt and implement directives based on personal business policies, and to adapt themselves to changing environments. Before using the MOAT system, several slowed down decisions impacted on the supply chain business, because the traceability system was configured with common basic policies, which often were not adequate. This limit often required the intervention of developers to hardcode specific constraints into the *SUs*. Finally, the exchange of information and the data protection have become more efficient by using tracing agents, because many business applications can now be connected to the network via different *AUs*. In the MOAT system, an *SU* can be easily adapted to a different workload by locally changing the policy, or by configuring a policy which takes the workload itself into account.

8. Conclusion

In this paper, we have proposed a solution for chain traceability that relies on agent-based and ubiquitous computing technologies. After a business and technological overview, encompassing traceability patterns, enterprise integration approaches, automatic identification and data capture technologies, the study has focused on the service-oriented composition drawbacks, promoting the multi-agent cooperation and self-adaptation to gain a better sustainability. Hence, key properties of agent-based traceability have been pointed out, giving a business process traceability model, an architectural vision, and then the behavioral model for both tracing and interface agents. Finally, the application of the system to a real-world case study has been discussed, pointing out its effectiveness and the benefits in terms of business process analysis.

Acknowledgements

This work was partially supported by the projects INNO.PRO.MODA, VIRGOAL and TRA.S.P, under the DOCUP OBJECTIVE 2, i.e., a research, innovation and development program hold by the Tuscany Region with European funds, for the development of the economic and productive regional factories. The authors thank the "100% Italiano" and ASSA consortiums, and the small enterprise Confelettronica s.r.l., for the valuable help in process modeling and system experimentation.

References

[1] Auto-ID Labs, <http://www.autoidlabs.org>, (accessed 2009).
[2] W.E. Barlow Jr., M.T. Brannon, J.L. Jiannuzzi, Radio frequency identification (RFID) technology at DELL computer corporation, in: Proceedings of the Southern Association for Information Systems Conference, Florida, USA, March 11–12, 2006, pp. 187–190.
[3] A. Bechini, M.G.C.A. Cimino, F. Marcelloni, A. Tomasi, Patterns and technologies for enabling supply chain traceability through collaborative *e*-business, Information and Software Technology 50 (4) (2008) 342–359.
[4] M. Bhattacharya, C.-H. Chu, T. Mullen. A comparative analysis of RFID adoption in retail and manufacturing sectors, in: Proceedings of the IEEE International Conference on RFID (RFID'08), Las Vegas, Nevada, USA, April 16–17, 2008, pp. 241–249.
[5] M. Bevilacqua, F.E. Ciarapica, G. Giacchetta, Business process reengineering of a supply chain and a traceability system: a case study, Journal of Food Engineering 93 (1) (2009) 13–22.
[6] M. Bertolini, M. Bevilacqua, R. Massini, FMECA approach to product traceability in the food industry, Food Control 17 (2) (2006) 137–145.
[7] G.A. Blissett, Establishing Trust Through Traceability, IBM Institute for Business Value, 2007.
[8] H. Caituiro-Monge, M. Rodriguez-Martinez, AWS-Net Traveler: autonomic web services framework for autonomic business processes, in: Proceedings of the 2005 IEEE International Conference on Services Computing (SCC'05), July 11–15, 2005, Orlando Florida, USA, pp. 270–272.
[9] M.L. Cheng, J.E.L. Simmons, Traceability in manufacturing systems, International Journal of Operations and Production Management 14 (10) (1994) 4–16.
[10] M. De Blasi, V. Mighali, L. Patrono, M.L. Stefanizzi, Performance evaluation of UHF RFID tags in the pharmaceutical supply chain, in: Proceedings of The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications (Tyrrhenian'09), 2–4 September, 2009, Pula, Italy, pp. 283–292.
[11] P. De Meo, A. Garrob, G. Terracina, D. Ursino, Personalizing learning programs with X-Learn an XML-based user-device adaptive multi-agent system, Information Sciences 177 (8) (2007) 1729–1770.
[12] K.A.-M. Donnelly, K.M. Karlsen, P. Olsen, The importance of transformations for traceability – A case study of lamb and lamb products, Meat Science 83 (2009) 68–73.

[13] C.A. Ellis, K. Kim, Process aware information systems: a human centered perspective, in: G. Dong et al. (Eds.), Asia-Pacific Web Conference/International Conference on Web-Age Information Management (APWeb/WAIM) '07, LNCS 4505, Springer-Verlag, Berlin Heidelberg, 2007, pp. 39–49.
[14] EPC Global, <http://www.epcglobalinc.org> (accessed 2009).
[15] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall, New Jersey, USA, 2007.
[16] FIPA, Foundation for Intelligent Physical Agents, <http://www.fipa.org>.
[17] FIPA, Contract Net Interaction Protocol Specification, 2002.
[18] FIPA, Abstract Architecture Specification, 2002.
[19] C. Floerkemeier, M. Lampe, Issues with RFID usage in ubiquitous computing applications, LNCS, Pervasive Computing, vol. 3001, Springer-Verlag, Berlin Heidelberg, 2004. pp. 188–193.
[20] F. Gandino, B. Montrucchio, M. Rebaudengo, E.R. Sanchez, On improving automation by integrating RFID in the traceability management of the agri-food sector, IEEE Transactions on Industrial Electronics 56 (2009) 2357–2365.
[21] C. Gershenson, F. Heylighen, When can we call a system self-organizing?, in: W. Banzhaf, T. Christaller, P. Dittrich, J.T. Kim, J. Ziegler (Eds.), Advances in Artificial Life, Seventh European Conference (ECAL'03), LNAI 2801, Springer, Dortmund, Germany, 2003, pp. 606–614.
[22] O.P. Günther, W. Kletti, U. Kubach, RFID in Manufacturing, Springer-Verlag, Berlin Heidelberg, 2008.
[23] Q. Guo, M. Zhang, A novel approach for multi-agent-based intelligent manufacturing system, Information Sciences 179 (18) (2009) 3079–3090.
[24] GS1 Traceability, <http://www.gs1.org/traceability> (accessed 2009).
[25] F. Heylighen, C. Gershenson, The meaning of self-organization in computing, in: IEEE Intelligent Systems, Section Trends & Controversies - Self-organization and Information Systems, July/August 2003, pp. 72–75.
[26] G. Hohpe, B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley, Boston, USA, 2004.
[27] International Telecommunication Union, The Internet of Things, ITU Internet Reports, Geneva, Switzerland, 2005.
[28] ISO, International Organization for Standardization, ISO 8402:1994, Quality management and quality assurance, <http://www.iso.org>.
[29] ISO, International Organization for Standardization, ISO 9001:2000 and ISO 9001:2008, Quality management systems – Requirements, <http://www.iso.org>.
[30] L. Ivantysynova, M. Klafft, H. Ziekow, O. Günther, S. Kara, RFID in manufacturing: the investment decision, in: Proceedings of Pacific-Asia Conference on Information Systems (PACIS'09), Hyderabad, India, July 10–12, 2009, Paper No. 41.
[31] M.H. Jansen-Vullers, C.A. van Dorp, A.J.M. Beulens, Managing traceability information in manufacture, International Journal of Information Management 23 (5) (2003) 395–413.
[32] N.R. Jennings, M. Wooldridge, Applications of agent technology, in: N.R. Jennings, M. Wooldridge (Eds.), Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, Berlin, Germany, 1998.
[33] G. Jianga, B. Hu, Y. Wang, Agent-based simulation of competitive and collaborative mechanisms for mobile service chains, Information Sciences 180 (2) (2010) 225–240.
[34] S. Keegan, G.M.P. O'Hare, M.J. O'Grady, Easishop: Ambient intelligence assists everyday shopping, Information Sciences 178 (2008) 588–611.
[35] L. Ke-Xing, F. Yu-Qiang, E-Commerce oriented automated negotiation based on FIPA interaction protocol specification, in: Proceedings of the Sixth International Conference on Machine Learning and Cybernetics (ICMLC'07), Hong Kong, China, Aug. 19–22, 2007, pp. 3199–3204.
[36] P. Lin, A. MacArthur, J. Leaney, Defining autonomic computing: a software engineering perspective, in: Proceedings of the Australian Software Engineering Conference (ASWEC'05), 2005, pp. 88–97.
[37] D.S. Linthicum, Next Generation Application Integration: From Simple Information to Web Services, Addison-Wesley Professional, Pearson Education Canada, 2003.
[38] Microsoft Corp., Distributed Component Object Model (DCOM) Remote Protocol Specification, <http://msdn.microsoft.com/en-us/library/cc226801>(PROT.13).aspx (accessed 2009).
[39] T. Moe, Perspectives on traceability in food manufacture, Food Science and Technology 9 (1998) 211–214.
[40] A. Mouseavi, A. Sarhadi, A. Lenk, S. Fawcett, Tracking and traceability in the meat processing industry: a solution, British Food Journal 104 (1) (2002) 7–19.
[41] L.U. Opara, Traceability in agriculture and food supply cha, in: A review of basic concepts, technological implications, and future prospects, Journal of Food, Agriculture and Environment 1 (1) (2003) 101–106.
[42] M.J. O'Grady, G.M.P. O'Hare, Mobile devices and intelligent agents – towards a new generation of applications and services, Information Sciences 171 (4) (2005) 335–353.
[43] D.E. O'Leary, Supporting decisions in real-time enterprises: autonomic supply chain systems, Information Systems for E-Business Management 6 (2008) 239–255.
[44] Object Management Group, Business Process Modeling Notation (BPMN) Specification, <http://www.bpmn.org> (accessed 2009).
[45] Object Management Group, Common Object Request Broker Architecture (CORBA) Specification, http://www.corba.org (accessed 2009).
[46] A. Poggi, M. Tomaiuolo, P. Turci, An agent-based service oriented architecture, in: Proceedings of the Eighth Workshop from Objects to Agents (WOA'07) Genova, Italy, Sept. 24–25, 2007, pp. 157–165.
[47] A. Regattieri, M. Gamberi, R. Manzini, Traceability of food products: general framework and experimental evidence, Journal of Food Engineering 81 (2007) 347–356.
[48] J.A. Rodrigues, P.C.L. Monteiro, J.Oliveria, J.M. de Souza, G. Zimbrão, Towards an autonomic enterprise: from autonomic business processes to autonomic balanced scorecard, in: Proceedings of the first Brazilian Workshop on Business Process Management (WBPM'07), Oct. 21–24, 2007, Gramado, RS, Brazil.
[49] D. Ronzani, The battle of concepts: ubiquitous computing, pervasive computing and ambient intelligence in mass media, Ubiquitous Computing and Communication Journal 4 (2) (2009).
[50] G. Roussos, Enabling RFID in retail, IEEE Computer 39 (3) (2006) 25–30.
[51] S.J. Russel, P. Norvig, Artificial Intelligence, second ed., A Modern Approach, Prentice Hall, NJ, USA, 2003.
[52] J. Ryu, D. Taillard, GS1 Global Traceability Standard, Business Process and System Requirements for Full Chain Traceability, Issue 1, GS1 Standard Document, 2007.
[53] S. Sasazaki, K. Itoh, S. Arimitsu, T. Imada, A. Takasuga, H. Nagaishi, S. Takano, H. Mannen, S. Tsuji, Development of breed identification markers derived from AFLP in beef cattle, Meat Science 67 (2004) 275–280.
[54] SOAP, Simple Object Access Protocol, <http://www.w3.org/TR/soap/> (accessed 2009).
[55] T. Staake, F. Thiesse, Extending the EPC network – the potential of RFID in anti-counterfeiting, in: Proceedings of the ACM Symposium on Applied Computing (SAC'05), Santa Fe, New Mexico, USA, March 13–17, 2005, pp. 1607–1612.
[56] M. Tu, L. Jia-Hong, C. Ruey-Shun, X. Kai-Ying, J. Jung-Sing, Agent-based control framework for mass customization Manufacturing with UHF RFID technology, IEEE Systems Journal 3 (3) (2009) 343–359.
[57] M.J. Uddin, M.I. Ibrahimy, M.B.I. Reaz, A.N. Nordin, Design and application of radio frequency identification systems, European Journal of Scientific Research 33 (3) (2009) 438–453.
[58] UN/EDIFACT, United Nations Electronic Data Interchange for Administration, Commerce and Transport <http://www.unece.org/trade/untdid/welcome.htm> (accesed 2009).
[59] W.M.P. van der Aalst, A.H.M. ter Hofstede, Mathias Weske, Business process management: a survey, in: W.M.P. van der Aalst et al. (Eds.), Business Process Management (BPM'03), LNCS 2678, Springer-Verlag, Berlin Heidelberg, 2003, pp. 1–12.
[60] W3C, Web Services Activity, <http://www.w3.org/2002/ws/> (accessed 2009).
[61] W3C, Web services description language version 2.0 (WSDL 2.0), Part 1: Core Language (accessed 2009).

[62] X. Wang, D. Li, Value added on food traceability: a supply chain management approach, in: IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI'06), Shanghai, China, June 21–23, 2006, pp.493–498.
[63] S.A. White, Introduction to BPMN, Business Process Trends (2004).
[64] D. Woods, T. Mattern, Enterprise SOA: Designing IT for Business Innovation, O'Reilly Media, Sebastopol, CA, 2006.
[65] M. Wooldridge, Agent-based computing, Interoperable Communication Networks 1 (1) (1998) 71–97.
[66] J. Zhang, P. Feng, Z. Wu, D. Yu, Automatic identification-enabled traceability in supply chain management, in Proceedings of 4th International Conference on Wireless Communications, Networking and Mobile Computing, (WiCOM'08), Dalian, China, October 12–14. 2008, pp.1–4.