Agent Interaction Modeling based on Product-Centric Data: A Formal Method to Improve Enterprise Interoperability

Marco Stuit and Gerben G. Meyer

Department of Business & ICT, Faculty of Economics and Business,
University of Groningen, Landleven 5, P.O. Box 800,
9700 AV Groningen, The Netherlands, +31 50 363 {7083 / 7194}
{m.stuit, g.g.meyer}@rug.nl

Abstract. This paper shows how companies can use product-centric data to locally represent the interactions with and between their partners. An agent-based interaction modeling language is used to capture these interactions graphically and formally. Moreover, a formal method is introduced that enables partners to automatically construct a global interaction diagram from their local interaction representations. This global interaction diagram improves enterprise interoperability, since it increases overall process visibility. A simple process example is used to illustrate the approach.

Keywords: agents, interaction structuring, global and local process representations, product-centric data, enterprise interoperability, business process modeling.

1 Introduction

Nowadays, there is an ongoing trend towards collaborative business processes that are provided by autonomous partners, which together deliver complex products and/or services in a collaborative network. This is explicitly shown by the emergence of new organizational forms like the networked or virtual enterprise [18]. The decentralized nature of cross-organizational business settings is a good match for agent technology [28, 35]. The autonomous partners in a collaborative business process can be seen as agents that execute certain tasks locally, while the interactions between the partners can be seen as interactions in a multi-agent system. To capture these interactions, the TALL modeling language has been developed [32, 33] (see Section 2). As TALL is interaction-based, it is suitable for creating a description of the multiple interactions that occur in cross-organizational business environments. In this paper, it is shown how these descriptions are built using TALL Interaction Structure (IS) diagrams.

The manufacturing industry is increasingly moving from a supplier-driven to a customer-driven market. This transition is a great challenge to the manufacturing process itself since it must be more flexible and robust as well as demonstrate enhanced scalability [4]. Supply chains are evolving in order to keep in line with these

developments. This is particularly shown by the move from conventional location-centric supply chains to product-centric supply chains [15], as shown in Figure 1. A conventional supply chain is based on a systems design that is focused on location-specific material accounts and transactions between locations. A solution-design that tracks and controls individual products independently of the location and the ownership of the product individual characterizes a product-centric supply chain. In [6], it is mentioned that product-centric integration makes explicit the role of the product as the coordinating entity in the delivery of customized products and services.

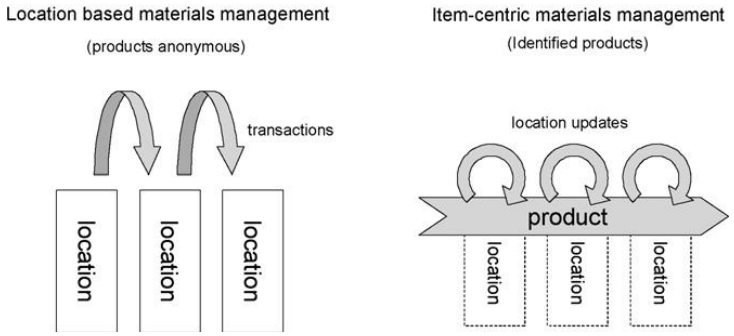

Fig. 1. Location-centric (left) and product-centric supply chains (right). Source [16].

Product-centric control is the focus of the TraSer project[1], in which control efforts are directed at individual products of the value-adding network. The idea behind product-centric control is that individual products and components are the basic entities in the information system rather than orders, production orders, or shipment batches. In the TraSer approach, each product is labeled with an RFID or barcode, containing an ID@URI code [17]. With this ID@URI, each product carries a unique identification number (ID), as well as a reference (Uniform Resource Identifier) to the location of the database where the product-centric data is stored. In this way, each partner in the supply chain can access this data, which results in product visibility throughout the supply chain. A recent survey about product-centric control and product visibility, as well as their enabling technologies can be found in [24].

The product visibility offered by the use of product-centric control does not imply process visibility. In this paper, it is proposed to use product-centric data to identify the multiple interactions between the different partners in the supply network, which form the collaborative business process. Currently, none of the partners has a global view of the entire collaborative business process in the supply chain, as each partner only has data about the interactions with direct partners. To increase interoperability between the partners, it is considered beneficial to build a global process definition. By using the product-centric data available in the network, each partner is enabled to identify the local interactions of and between all the partners in the supply chain. The identified local interactions can be merged to create a global process definition. This global process definition increases the understandability and visibility of the supply

[1] http://www.traser-project.eu

chain for this partner. In this paper, an algorithm is presented that enables partners to automatically create a global process definition from identified local interactions of partners. The goal of the algorithm is to build a minimal global interaction representation where the local interactions are merged when there is no conflict. A conflict indicates the existence of alternative local views on the same interaction. In the remainder of this paper, this algorithm is called the global construction algorithm. The approach presented in this paper assumes a common shared ontology in the business domain under consideration. Ontology matching is outside the scope of this paper.

The paper is structured as follows. Section 2 introduces the TALL modeling language and covers related work. The language is applied to a case example in Section 3. Next, Section 4 presents the global construction algorithm that enables partners to merge different local interaction representations. Section 5 applies this algorithm to the case example. After, Section 6 addresses public and private process representations in the context of TALL. A discussion is presented in Section 7 along with directions for future research. The paper ends with conclusions in Section 8.

2 Agent Interaction Modeling

This section discusses agent interaction modeling. Section 2.1 introduces the TALL modeling language. Afterwards, Section 2.2 motivates and evaluates TALL against related modeling languages.

2.1 The TALL Modeling Language

A network of organizations (e.g. a supply chain) can be viewed as a multi-agent system consisting of autonomous agents that are involved in the provision of a collaborative business process. Therefore, all the partners in a supply chain can be viewed as agents. In order to capture the interactions between these agents, a new graphical modeling language named TALL (The Agent Lab Language) has been developed [32, 33]. In the language, collaborative business processes are conceived as a structure of role-based interactions through which agents cooperate and coordinate their work. Similarly, related research considers business processes as a special kind of social interaction process [36] or as social constructs [21].

The top part of Figure 2 shows the abstract symbols that represent the essential components of the language. Rounded rectangles represent *agents*, ellipses represent *roles*, and hexagons represent *interactions*. From a modeling perspective, TALL distinguishes between atomic agents (human and software agents) and composite agents (synthetic agent). Graphically, icons that appear in the top left corner of the agent symbol distinguish between the different types of agents. Circle icons represent human agents, square icons represent existing software agents, and triangle icons represent synthetic agents. By playing roles, agents interact with other agents. Moreover, Figure 2 demonstrates how the symbols are connected to denote that agents play the roles that are attached to an interaction. The fact that an agent is

playing a role means that the agent has the necessary skills and responsibilities to perform the required activities.

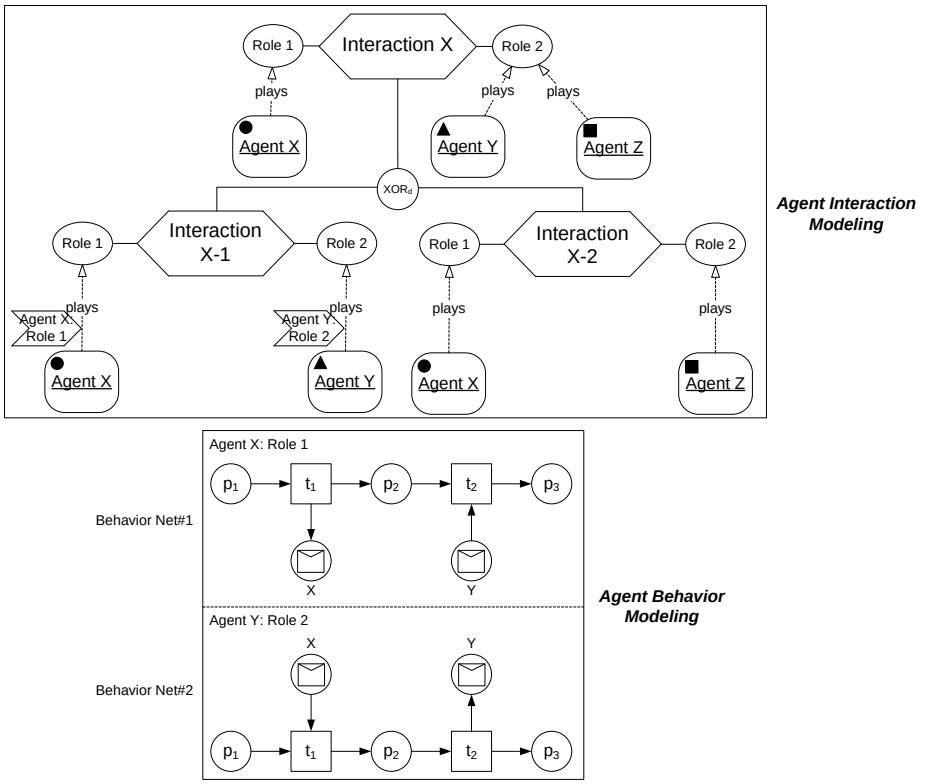

Fig. 2. The separation between agent interaction and behavior modeling in TALL. The Interaction Structure (IS) diagram shows how a collaborative business process is represented as an interaction tree with agents playing the roles attached to the interactions. Agents bring explicit behaviors in the form of Agent Behavior (AB) diagrams to perform interactions.

The different agents interact autonomously in order to complete their individual and joint goals. Autonomy means here that the participants act in empowered roles in which they can make decisions and take actions on their own. Interactions are related to other interactions through composition (one interaction being part of another) and dependency (one interaction must be completed before the other can start). The Interaction Structure (IS) diagram depicts interactions and their relations in a tree structure. The sub-interactions or direct child interactions are linked to their parent interaction by a routing symbol that indicates the process flow of the children. The parent interaction completes if all sub-interactions have completed taking into account the specified routing.

In Figure 2, the XOR_d label indicates an exclusive deterministic OR-split, that is, only one of the two sub-interactions is executed. If either interaction *X-1* or interaction *X-2* completes, interaction *X* completes. Thus, completion is a bottom-up

process in the IS diagram. Besides XOR, also sequential (SEQ) and parallel (PAR) routing is supported by the language. Graphically, the (sequential) order of a set of children under a common parent is read from left to right. A decision rule set is assigned to the three routing types to enable condition-based execution of a set of direct child interactions. Graphically, a (non-empty) decision rule set is depicted by attaching a subscripted letter *d* to the routing type (as in Figure 2). Decision rule sets can for instance be used to achieve an exclusive deterministic choice (XOR_d), or to make an inclusive choice for N-out-of-M direct children. In the latter case, when $N \geq 2$, the children are executed in sequence (SEQ_d) or in parallel (PAR_d).

With TALL, collaborative business processes are addressed in two ways. A clear separation exists between (1) the composite interaction specification in the IS diagram and (2) the specification of the agent's behaviors. The IS diagram introduces a higher-level notion of process that focuses on inter-agent interactions whereas behavioral models focus on individual agent behaviors. The agents' behaviors are modeled using swimlaned process diagrams named TALL Agent Behavior (AB) diagrams. AB diagrams are based on the Behavior net formalism [22], which is an extension of the colored Petri net formalism with the message place type. An agent that is assigned to a specific role in the IS diagram owns an AB diagram that is used to perform the specific interaction. The bottom part of Figure 2 shows an example in which *Agent X* and *Agent Y* own an AB diagram that is used to perform interaction *X-1*. Each AB diagram represents the internal states of the agent and the activities/events that cause the agent to change states. Message places enable the flow of tokens between different agent behaviors in an interaction.

In the IS diagram, AB diagrams appear on a higher level of abstraction in the form of chevron symbols. A chevron symbol carries an agent-role label and is connected to the tail of the agent-role connector (as in Figure 2). In this way, a chevron symbol appears adjacent to its owner agent. Each chevron symbol serves as an abstract representation of the agent's local behavior in the form of an AB diagram. References [22] and [23] discuss in more detail the agent behaviors and how nonaligned behaviors are dealt with. In nonaligned behaviors, the message places are not necessarily matched by type name and number like in Figure 2.

The IS diagram is mainly used to increase process understandability and visibility for analysis and/or redesign purposes. However, it can also support the execution of the process by acting as a coordination structure for an (agent-based) execution environment [29]. Such an environment performs the collaborative business process by coordinating the distributed agent behaviors. In this paper, the focus is on the IS diagram as a tool for supply chain partners to automatically build a global interaction representation from local interaction representations.

2.2 Related Work and Motivation

In this section, TALL is compared to existing agent-based modeling techniques and traditional process modeling languages.

Traditional process modeling languages like BPMN [37], UML [12], EPCs [30], IDEF0 [20], and Petri nets [11] create structured well-defined process models. In these languages, the collection of tasks or activities in a business process is ordered in

a prescriptive process model that is performed in an imperative way [27]. Such languages are appropriate for modeling business processes that display complex flows, but are less appropriate for modeling business processes that involve the co-operation of several entities [21].

Contemporary organizations are complex collaborative networks [7]. The business processes of such organizations involve more people that collaborate both within and across organizations. In such business processes, collaboration instead of task sequence determines the nature of the working activity. According to [14], collaborative activity simply does not match in any way the underlying "parallel flowchart" paradigm of current mainstream process notations and languages. The development of TALL is motivated by the need for new methods and languages to model collaborative work practice.

TALL is inspired by the agent paradigm to model collaborative work practice. Existing agent-based languages include (amongst others) stand-alone languages like AUML [2], AORML [36], AML [8], and OPM/MAS [34]. Moreover, agent-based languages are at the centre of well-known agent-oriented software engineering methodologies like MASE [10], Prometheus [26], TROPOS [3], GAIA [38], and MESSAGE [5]. Although many of these languages incorporate the same basic agent-oriented constructs, they are mainly focused on the software development process for agent systems. These languages are software design techniques that are mainly used by software architects. Hence, they do not focus on business process specification.

The main contribution TALL makes as a modeling language is that it presents a process-oriented approach, based on agent-oriented concepts and notations, which is more useful for business architects and analysts. The language focuses on process modeling, and can be used outside a software development or system implementation context. TALL should be seen as a complement to the existing agent-based languages and frameworks that have proven to be useful for agent-based software development activities. The main strength of the language is the strong conceptual separation between global agent interaction specification and local agent behavior specification. The language is based on the premise that human collaborative work is not about carrying out steps in a pre-defined sequence, but instead requires a higher-level notion of process. Therefore, the IS diagram allows general constraints to be put on the overall process independently of the agent behaviors. In other words, process-wide behavior can be partially defined at design time and used at run-time independently of the autonomous behaviors of the agents.

The agent behavior specification is done from a local viewpoint using explicit process models in the form of AB diagrams. The use of explicit process models is in line with a process-based approach. In comparison, in traditional process modeling languages the concept of locality does not exist. Local process variation is lost at the whim of the business analyst who creates a centralistic high-level model of the process in order to increase process standardization. TALL models the operational flow of the process from the local perspectives or beliefs of the agents [31]. In this way, TALL is also in line with an agent-based approach. For a more elaborate discussion on TALL's modeling concepts and notations, and an explicit comparison with the related modeling languages above and several others the reader is referred to [32, 33].

3 Product-Centric Case Example

This section presents a typical tracking and tracing case from the perspective of one agent in a supply chain. This scenario is used to illustrate how this one agent can build a global process definition of the supply chain using local process information (i.e. interactions this agent has with its direct partner agents) and product-centric data, which is linked to individual products. The scenario is a simplified supply chain, based on [9], in which agents have the shared goal of delivering Personal Computers (PCs) to end customers. The scenario is shown in Figure 3.

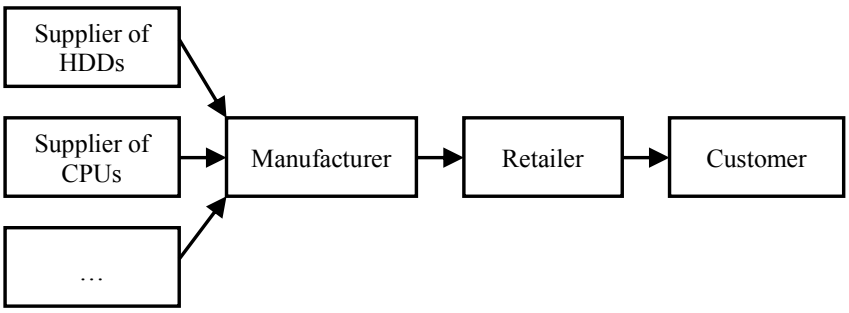

Fig. 3. The simplified supply chain.

In this supply chain, several supplier agents deliver computer parts to a manufacturer agent, who assembles the computer(s), and ships them to a retailer agent. The retailer agent sells the assembled computer(s) to a customer. This scenario is focused on the view of the retailer. The retailer can build an IS diagram of the interactions he has with his direct partner agents. This diagram is shown in Figure 4. The roles are omitted in this diagram for clarity reasons. However, the agents are playing their specific roles as supplier, manufacturer, retailer, or customer. As can be seen in the diagram, the root interaction of the retailer agent is “Deliver PCs”, which is also the common shared goal of the whole supply chain. For this purpose, the retailer interacts with the manufacturer (called “Manufacturer-Retailer”) and with the customer (called “Retailer-Customer”). In case of the interaction with the manufacturer, the retailer agent interacts with him to order PCs, and later on to receive the shipments of PCs. The interaction with the customer is a bit more complex, it consists of a partially ordered set of interactions, in which a quote is requested, a quote is send to the customer, and if the customer accepts the quote, an order and shipment of the PCs follows.

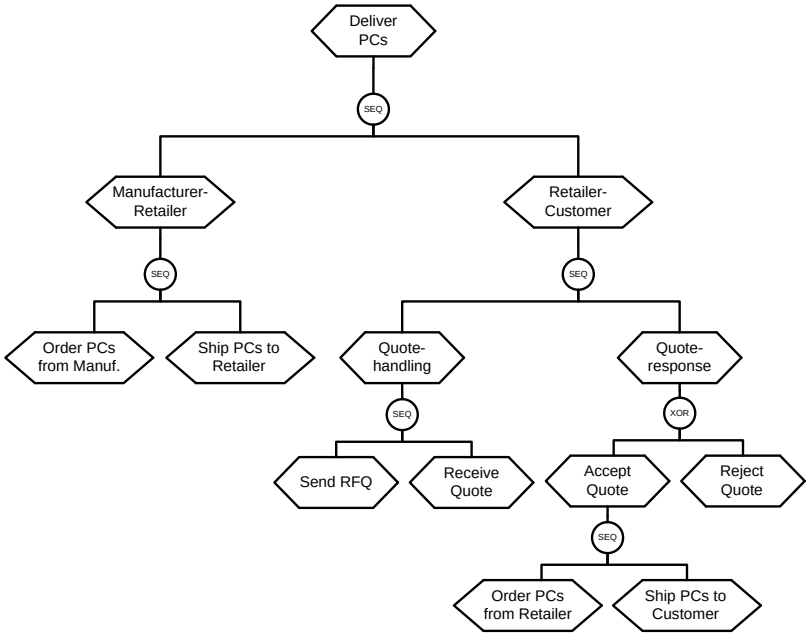

Fig. 4. The IS diagram of the retailer.

To improve the insight of the retailer agent in the whole supply chain, the retailer agent can build IS diagrams from the perspective of the other agents in the supply chain by generalizing over available product-centric data. Table 1 shows an example of product-centric data of an individual product. However, the product-centric data only reveals a part of the interactions in which the other agents are involved, as the product-centric data per product individual only contains information about location and shape changes including the time when these changes happened. Based on this, the proposed approach assumes incomplete interaction models.

Table 1. Example of product-centric data.

Product-ID	Time	Type	Location
123456	10.01.2008 12:34	Shipment	Manufacturer X
123456	15.01.2008 13:35	Assemble	Manufacturer X
123456	20.01.2008 14:36	Shipment	Retailer Y
123456	25.01.2008 15:37	Shipment	Customer Z

The behaviors used by the partners within the interactions remain private knowledge, as this approach only reveals interactions with and between partners, which are extending the product-centric data. Furthermore, confidentiality is provided to partners as they can decide to either share or withhold certain product-centric data. The benefits of the proposed approach are maximized in situations where the trust between partners is high. In such situations, partners feel less inhibition to share

product-centric data (or any other information) throughout the supply chain. Trust between partners is usually built up in long-term supplier-customer relationships. In case of complex products, product-centric data of the components of the product can also be retrieved, at least if the data of the components is stored with or linked to the product data.

As mentioned before, a partial IS diagram of the manufacturer agent can be built by the retailer agent, based on the available product-centric data. Figure 5 illustrates this diagram for the described scenario. Such a diagram can be built by using a set of simple rules, based on the location changes of the product, as well as the physical changes made to the product. A physical change is modeled as an interaction, as it can be considered an interaction between the manufacturer and the product. An example is the "Assemble PCs"- interaction in Figure 5.

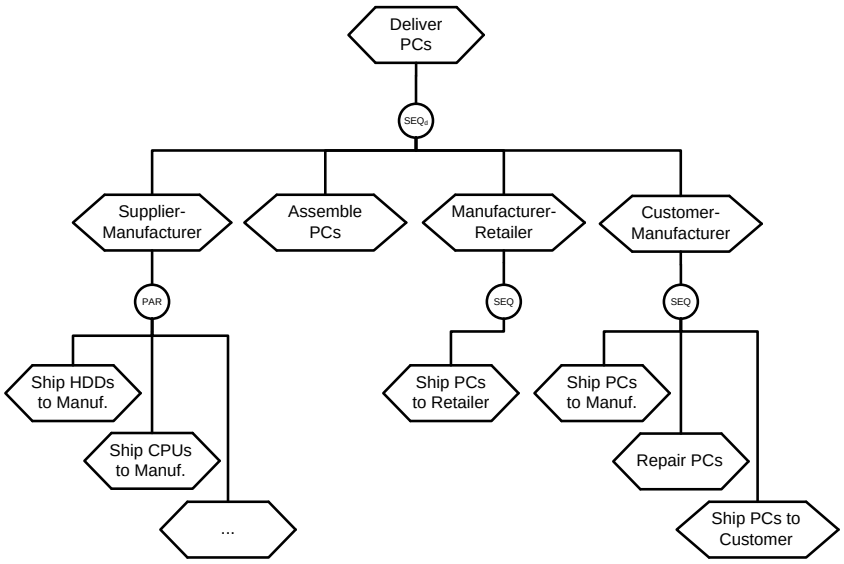

Fig. 5. The IS diagram of the manufacturer from the retailer's viewpoint.

This diagram only reveals when products are shipped between agents, when products are changed, and in which order this occurs. Therefore, an interaction is allowed to only have one child, since the interactions identified by using the product-centric data can be incomplete like for instance the "Manufacturer-Retailer"-interaction in Figure 5. The "Customer-Manufacturer"-interaction does not always occur, but only when one or more of the delivered computers need to be repaired by the manufacturer agent. Thus, a decision rule set that can check this condition is attached to the SEQ routing type that graphically appears below the "Deliver PCs"-interaction. If one or more of the computers needs to be repaired, all interactions occur in sequence. If not, N-out-of-M children occur in sequence. In this specific example, three out of four children occur in sequence in the latter case.

In a similar way, an IS diagram from the perspective of the customer agent can be created by the retailer agent. This diagram is shown in Figure 6. Again, by using the

product-centric data, only the shipments of and physical changes to the product are revealed.

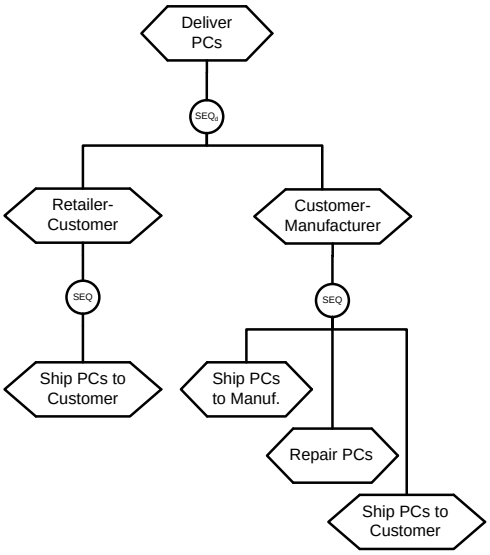

Fig. 6. The IS diagram of the customer from the retailer's viewpoint.

4 Building the Global Interaction Diagram

This section presents the method for building a global IS diagram from a set of local IS diagrams. Section 4.1 introduces the global construction algorithm. After, Section 4.2 discusses tool support.

4.1 Global Construction Algorithm

As explained, each agent involved in a collaborative business process can use product-centric data to identify agent-to-agent interactions. Each agent builds a set of local IS diagrams (in which each set can be different), representing its own interactions, as well as interactions other agents are performing. This is considered necessary, as agents (in a supply chain) are not always willing to exchange their process definitions. The global construction algorithm is listed in Table 2. The algorithm is expressed in pseudo-code, which uses set-theoretic notation mixed with the control structures of conventional high-level programming languages. Below Table 2, the different parts of the algorithm are explained.

Figure 7 shows the input and output of the algorithm graphically. The algorithm expects a global IS diagram *GI* (Global Input) which only contains the root interaction

i_0. The root interaction represents the common shared goal of the collaborative business process under consideration. Furthermore, the algorithm expects a set of local IS diagrams *LI* (Local Input) that are to be merged. Each local diagram is based on the same root interaction i_0. The algorithm returns a new global diagram *GO* (Global Output). As mentioned before, the goal of the global construction algorithm is to build a minimal interaction representation *GO* where the interactions, from the diagrams that are being compared, are merged when there is no conflict. Conflicts are detected by the function **CONFLICT** (see Table 3 and the explanation below it).

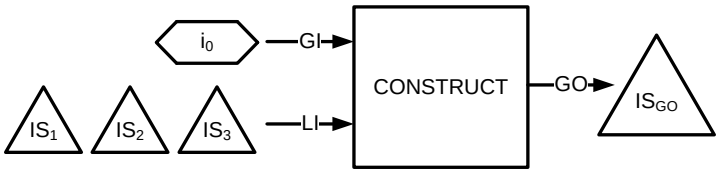

Fig. 7. Illustration of the global construction algorithm, as executed by every agent.

The global construction algorithm uses some notations from the formal definition of the TALL interaction diagrams [33]. Here, these notations are explained informally:

- *I* is the set of interactions that form an IS diagram;
- $L_I(i)$ is a function that assigns a label to each interaction from the set *I*. Although interactions can share labels in the same diagram, each interaction is unique. Thus, interactions with identical labels are completely unrelated when executed;
- *RT* is the routing type function that assigns a routing type to each interaction from the set *I*. The function *RT* indicates how the direct child interactions are routed in order to complete their parent interaction. As indicated above, the supported routing types are SEQ, PAR, and XOR;
- The partial ordering relation $<_i \subseteq I \times I$ connects parent interactions to their child interactions. Graphically, a line is drawn between a parent and each direct child. These lines converge in the routing type symbol that graphically appears below each parent;
- *DR(i)* is a function that assigns a decision rule set to each interaction. As mentioned above, graphically the routing type is augmented with a subscripted letter *d* if the decision rule set is non-empty;
- The set *R* contains all roles that are relevant to the set of interactions *I*;
- *RI* is a function that assigns roles to interactions. Graphically, this is depicted by a connector line between a role and an interaction (as in Figure 2);
- $L_R(r)$ is a function that assigns a label to each role from the set *R*;
- i_0 denotes the root interaction;
- *children(i)* is a function that returns the set of direct child interactions of the parent interaction *i*;
- *parent(i)* is a function that returns the parent of the interaction *i*;
- *ht(i)* is a function that returns the height of interaction *i*.

In addition, some new operators and notations are used in the global construction algorithm, as an extension of the formal definition in [33]:

1. an operator for the union of two IS diagrams is needed. If IS_1 and IS_2 are two IS diagrams, then the union of IS_1 and IS_2 is an IS diagram $IS_{IS1} \cup_{IS2} = IS_1 \cup IS_2$ such that: $I_{I1} \cup_{I2} = I_1 \cup I_2$, $<_{I <I1} \cup_{<I2} = <_{I1} \cup <_{I2}$, $R_{R1} \cup_{R2} = R_1 \cup R_2$, $RI_{RI1} \cup_{RI2} = R_{I1} \cup R_{I2}$. The union of the set of interactions and the set of roles includes the union of the interaction attributes (label, routing type and decision rule set) and role attributes (label). Hence, if I_1 and I_2 are two sets of interactions then the union of I_1 and I_2 is an interaction set $I_{I1} \cup_{I2} = I_1 \cup I_2$ such that: $L_{I1} \cup_{I2} = L_{I1} \cup L_{I2}$, $RT_{RT1} \cup_{RT2} = RT_1 \cup RT_2$, $DR_{DR1} \cup_{DR2} = DR_1 \cup DR_2$. Similarly, if R_1 and R_2 are two sets of roles then the union of R_1 and R_2 is a role set $R_{R1} \cup_{R2} = R_1 \cup R_2$ such that: $L_{R1} \cup_{R2} = L_{R1} \cup L_{R2}$;
2. *subtree(i)* is a function that returns a segment or subtree of the IS diagram with root *i*. Thus, *subtree(i)* $\subseteq$ *IS* (see Figure 8);

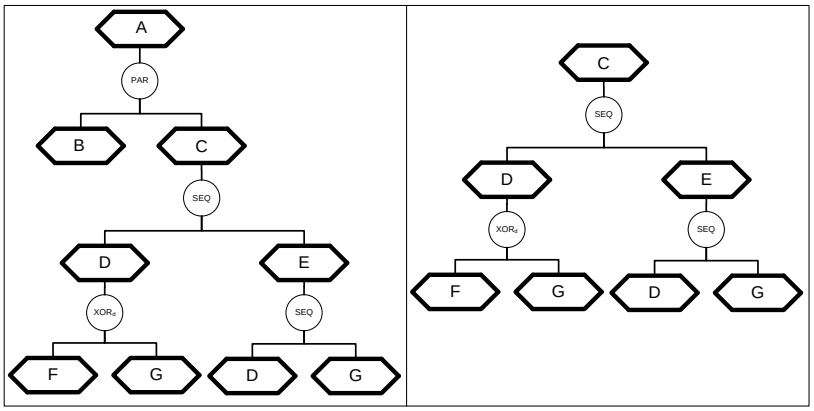

Fig. 8. Illustration of the input (left) and output (right) of the *subtree* function applied to interaction *C*.

3. *comp(i)* is a function that returns a single component of the IS diagram. In other words, it returns a subtree (read: diagram) with the single interaction *i*. In this regard, a component comprises an interaction, and all its attributes and roles;
4. *clone(i)* is a function that returns a new interaction with all attributes and roles of *i*. In other words, it creates a copy of the component *i*;
5. *dum:* $I \rightarrow \{TRUE, FALSE\}$ is a function that is used to differentiate between 'normal' interactions and dummy interactions. Dummy interactions are introduced during execution of the global construction algorithm to act as the parent of identically labeled interactions, which the algorithm cannot merge because of a conflict. Such interactions are alternative views on the same interaction. This means all children of a dummy interaction have identical labels. Initially, the default setting for all interactions is FALSE: $\forall$ i $\in$ I: $\neg$ dum(i);
6. *IH* is an ordered set in which all interactions $i \in I$ are ordered according to the binary relationship $R = \{(i_1, i_2) \mid i_1 \in I \wedge i_2 \in I \wedge ht(i_1) \leq ht(i_2)\}$;

7. elements are augmented with an index *GI, GO,* or *li* indicating that the element appears in the global input diagram, the global output diagram or in the local diagram under consideration. For instance, interaction i_0^{GO} is the root interaction that appears in the global output diagram.

Table 2. Global Construction Algorithm.

```
 1: function CONSTRUCT (GI,LI)
 2: begin
 3:       GO := GI
 4:       for each li ∈ LI do
 5:             for each i ∈ IH^li do
 6:                   if i ∉ subtree(parent(i)^li)^GO then
 7:                         i_new := clone(i)^li
 8:                         GO := GO ∪ {comp(i_new)}
 9:                         <_I^GO := <_I^GO ∪ {(parent(i)^li, i_new)}
10:                   else if conflict(subtree(i)^GO, subtree(i)^li) then
11:                         i_dum := j: j ∈ I^GO ∧ i = j ∧ dum(j)
12:                         if ¬∃ i_dum then
13:                               i_dum := clone(i)^GO
14:                               DR(i_dum) := ∅
15:                               dum(i_dum) := TRUE
16:                               GO := GO ∪ {comp(i_dum)}
17:                               <_I^GO := <_I^GO ∪ {(parent(i)^GO, i_dum)}
18:                               <_I^GO := <_I^GO \ {(parent(i)^GO, i^GO)}
19:                               <_I^GO := <_I^GO ∪ {(i_dum, i^GO)}
20:                         i_conflict := clone(i)^li
21:                         comp(i_dum)^GO := comp(i_dum)^GO ∪ comp(i_conflict)^GO
22:                         RT(i_dum) := XOR
23:                         GO := GO ∪ {comp(i_conflict)}
24:                         <_I^GO := <_I^GO ∪ {(i_dum, i_conflict)}
25:                   else
26:                         comp(i)^GO := comp(i)^GO ∪ comp(i)^li
27:       return GO
28: end
```

The global construction algorithm is contained into a main **for each**-loop (Lines 4-27) that for each iteration compares *GO* to a local IS diagram $li \in LI$. During the comparison, the algorithm searches for commonalities and/or differences between *GO* and *li* by comparing (labels of) the interactions in both diagrams. If possible, interactions in both diagrams are merged and a new global IS diagram is produced that serves as input for the next iteration. This global IS diagram is compared with the next local diagram from *LI*, until all the local IS diagrams have been processed. In the end, the global construction algorithm produces a global IS diagram *GO* that takes into account all the local IS diagrams of the agents involved.

The global construction algorithm first assigns *GI* to *GO* (Line 3). Inside the main **for each**-loop, an inner **for-each** loop (Lines 5-26) processes one by one all the

interactions from the local diagram *li* under consideration according to the ordering in the set *IH*. This means the algorithm starts with the root interaction from the local diagram *li* that has height zero. Next, the algorithm continues with the interactions that have height one (i.e. the direct child interactions of the root interaction), and so until all the interactions in the local diagram *li* have been processed. In general, three cases can apply to the interaction i^{li} that is being processed.

Case 1: Interaction i^{li} does not exist in the same segment of GO (Line 6)
The **if** statement in Line 6 of the algorithm tests whether interaction i^{li} should be added to *GO*. Interaction i^{li} is added to *GO* when i^{li} is does not exist in the subtree of its parent in *GO*. When the condition in Line 6 is true, a clone of i^{li} is created (Line 7), this clone is added to *GO* (Line 8) and connected to the correct parent in *GO* (Line 9).

Figure 9 shows a generic example (without roles) in which the condition in the **if** statement is true for interaction D^{li}. In the specific example, interaction D^{li} does not exist in the subtree of its parent in *GO*, that is, the subtree of A^{GO}. Therefore, interaction D^{li} including its attributes and roles are added to *GO*. Each IS diagram shows a local process view, from one agent's perspective, that is build based on product-centric data. In this regard, each IS diagram represents a specific incomplete part of the overall collaborative business process in the supply chain. Thus, the local IS diagrams, which are input to the algorithm are assumed to be incomplete.

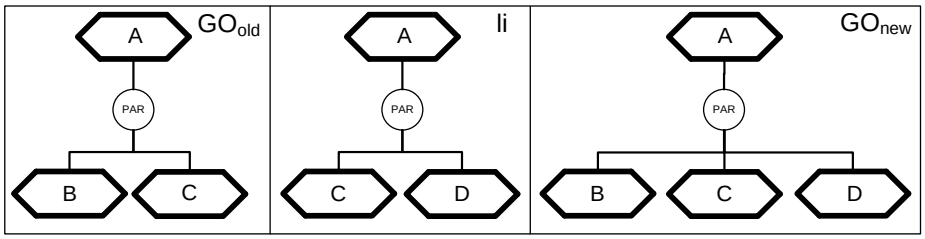

Fig. 9. A generic example of a Global Input diagram (GO_{old}), a local input diagram (li), and a Global Output diagram (GO_{new}) that illustrates the case in which interaction D^{li} is added to GO_{new} since it does not exist in GO_{old}.

In Figure 9, GO_{old} and *li* are the two IS diagrams being compared. Both diagrams show the incomplete local process perspective or view of an agent. The local IS diagram of the first agent shows that interactions *B* and *C* occur in parallel as part of interaction *A*. The local IS diagram of the second agent shows that interactions *C* and *D* occur in parallel as part of interaction *A*. The two agents provide complementary views that are merged in the output in order to create a more complete representation. Based on this, it is sufficient to perform interaction *A* once in GO_{new}. However, merging only occurs when the views of agents are not conflicting. Case 2 discusses how the algorithm deals with a conflict situation.

Case 2: Interaction i^{li} exists in the same segment of GO and generates a conflict situation (Line 10)

If the condition in Line 6 is false, which means i^{li} has a counterpart i^{GO} in the same segment, the algorithm continues with the **else if** statement in Line 10. The **else if** statement tests whether there is a conflict between the interactions i^{GO} and i^{li}. A conflict is detected by the function **CONFLICT** that is listed in Table 3 and explained below Table 3. In general, a conflict means that the interactions being compared cannot be merged since they are alternative views on the same interaction. Hence, the global construction algorithm keeps both interactions (i^{GO} and i^{li}) in the global output diagram. A new interaction i_{dum} (dummy interaction) is introduced (Line 13) to act as the parent of i^{GO} and i^{li}. Since i_{dum} carries the decision rule set of i^{GO} (assigned by the clone function in Line 13), Line 14 empties the decision rule set of i_{dum}. Next, Line 16 adds i_{dum} including all its attributes and roles to *GO*. The algorithm then assigns i_{dum} as parent interaction of i^{GO} (Line 19). In the scenario in which i^{GO} already has a parent, Lines 17 and 18 make sure that this parent is detached as parent of i^{GO} and instead is assigned as parent of i_{dum}. Any newly introduced dummy interaction is assigned TRUE by the function *dum(i)* (Line 15). Together with Lines 11 and 12 this prevents that in a future iteration another dummy interaction is introduced in *GO* for the same i^{GO} and i^{li}. Since i_{dum} acts as parent of i^{GO} and i^{li}, the next step is to add i^{li} to *GO*. To this end, Line 20 first creates a clone of i^{li} named $i_{conflict}$. Line 23 adds the clone of i^{li} to *GO* and Line 24 assigns this clone as a child of i_{dum}. Line 21 assigns the union of the components of i_{dum} and $i_{conflict}$ to the dummy interaction. In this way, the combined roles of its child interactions are attached to i_{dum}. Since the direct child interactions of i_{dum} are different views on the same interaction, i_{dum} is assigned the routing type XOR by Line 22. The use of a different routing type than XOR would imply that the interaction is executed more than once.

Table 3. The function **CONFLICT**.

```
 1: function CONFLICT (sGO,sli)
 2: begin
 3:       if RT(i0^GO)^GO ≠ RT(i0^li)^li then
 4:             return TRUE
 5:       else
 6:             X := children(i0^li)^li
 7:             Y := X ∩ GO
 8:             for each j ∈ Y do
 9:                   if ht(j)^GO ≠ ht(j)^li then
10:                         return TRUE
11:       return FALSE
12: end
```

The function **CONFLICT** receives as input, from the global construction algorithm, the subtrees of i^{GO} and i^{li} (see Table 3). This means that the root interactions in the function **CONFLICT** are (the currently being processed) interaction i^{li} in the global construction algorithm and its counterpart i^{GO}. A conflict arises in two situations. First, when the routing types of i^{GO} and i^{li} are different (Line 3 in Table 3) they are considered alternative views on the same interaction. In this case,

the function returns TRUE (Line 4 in Table 3). Next, the function builds a set *X* that contains all children of i_0^{li} (Line 6 in Table 3) and builds a set *Y* that contains all interactions that are members of both *X* and the set of interactions in *sGO* (Line 7 in Table 3). For all the interactions in the set Y, the function then tests whether these interactions occur on the same level (i.e. height) in *sGO* and *sli*. In other words, the function checks whether the children of i_0^{li} have counterparts in *sGO* that occur another level. If this is not true, there is no conflict and the function returns FALSE (Line 11 in Table 3). However, if this is true (Line 9 in Table 3) then i_0^{GO} and i_0^{li} are considered alternative views on the same interaction and the function returns TRUE (Line 10 in Table 3). The latter forms the second conflict situation.

Figure 10 shows a generic example in which the interactions A^{GO} and A^{li} are in conflict because they have different routing types, that is, they are alternative views of the same interaction. A new (dummy) interaction with routing type *XOR* and the same label is introduced in *GO* to act as the parent of A^{GO} and A^{li}.

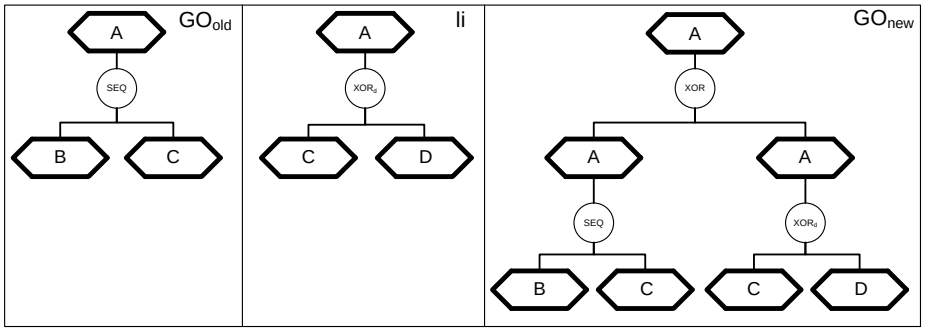

Fig. 10. A generic example of a Global Input diagram (GO_{old}), a local input diagram (li), and a Global Output diagram (GO_{new}). The example illustrates the case in which A^{li} generates a conflict because a counterpart A^{GO} exists that has a different routing type.

Figure 11 depicts a generic example of the second conflict situation. In this specific example, interactions A^{GO} and A^{li} are in conflict because parent interaction A^{li} has a child D^{li} with a counterpart D^{GO} that exists in the subtree of A^{GO} on a different level. Therefore, the same dummy interaction *A* with routing type *XOR* is added to *GO* in this situation.

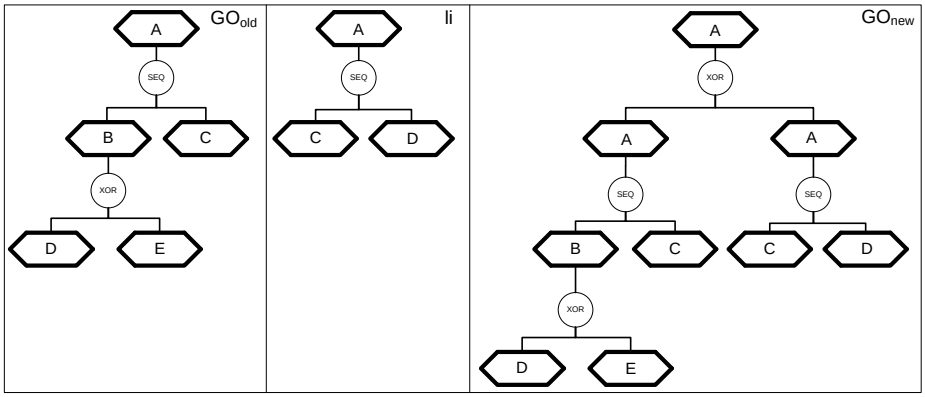

Fig. 11. A generic example of a Global Input diagram (GO_{old}), a local input diagram (li), and a Global Output diagram (GO_{new}). The example illustrates the case in which A^{li} generates a conflict because it has a child that occurs in the subtree of its counterpart A^{GO}.

Case 3: Interaction i^{li} exists in the same segment of GO and there is no conflict situation (Line 25)

From here, any line numbers refer to Table 2 again. If i^{li} exists in the same segment of *GO* but there is no conflict, the statement in Line 26 is executed. In this case, the components of i^{GO} and i^{li} are simply merged since it is possible for them to have different roles, decision rules etc. Figure 9 provides an example of this case. In Figure 9, parent interaction A^{li} has a child C^{li} with a counterpart C^{GO} in the subtree A^{GO}. However, in this case there is no conflict between A^{GO} and A^{li} because C^{li} and C^{GO} occur on the same level in GO_{old} and *li*. Thus, in Figure 9 it is unnecessary to keep both A^{GO} and A^{li} in the output by introducing a dummy interaction.

4.2 Tool Support

A software toolset is available to create and modify TALL diagrams, diagram elements, and properties of these elements using a graphical interface[2]. All diagrams, elements, and their properties are stored in an associated database. The current release of the software tool contains an implementation of the global construction algorithm. The user can create multiple IS diagrams and merge these into a global IS diagram.

[2] The latest release of the software tool can be downloaded from the software section on http://www.agentlab.nl/

5 Applying the Global Construction Algorithm

This section applies the global construction algorithm to the example case described in Section 3 in order to illustrate the workings of the algorithm. The retailer agent in the supply chain can run the algorithm, using the different IS diagrams it has generated (as shown in Figures 4, 5, and 6) as inputs. Figure 12 shows the generated output of the global construction algorithm. This global IS diagram shows the process starting from the supplier, until the product reaches the customer, and afterwards, when optionally a product is returned by the customer to be repaired by the manufacturer. Despite its simplicity, this example demonstrates several important features of the global construction algorithm.

First, this example shows how different views on an interaction are merged when an interaction i^{li} appears in the local diagram *li* under consideration but does not appear in the global diagram *GO*. This reflects the first case in the global construction algorithm. An example of such an interaction is the "Assemble PCs"-interaction from the manufacturer diagram (see Figure 5), which does not appear in the retailer diagram (see Figure 4). A clone of this interaction is created, the clone is added to *GO*, and connected to the same parent in *GO* (see Figure 12).

Second, this example shows how the algorithm deals with a conflict situation, that is, the second case in the algorithm. In the example, the retailer and the customer diagrams both include the "Retailer-Customer"-interaction. The "Ship PCs to Customer"-interaction, which is a child of the "Retailer-Customer"-interaction in the manufacturer diagram, exists also, on a different level, in the subtree of the "Retailer-Customer"-interaction in the retailer diagram. Because of this, the function **CONFLICT** returns TRUE since the two "Retailer-Customer"-interactions are considered alternative views on the same interaction. In this case, the global construction algorithm keeps both views in the output, and adds a new parent interaction for both views (see Figure 12). In this way, the ordering constraints of both views are preserved. Since both views are alternative descriptions on the same interaction, the new parent interaction is assigned the XOR routing type.

The third feature shown is how the algorithm deals with the third case in the algorithm. For instance, when the root interactions in the retailer and manufacturer diagrams are being compared, a conflict does not arise. These interactions have similar routing types and the children of the "Deliver PCs"-interaction in the manufacturer diagram do not occur on a different height in the retailer diagram. Thus, the **CONFLICT** function returns FALSE and the algorithm executes the statement in Line 26. This statement merges the attributes and roles of both interactions. This merging includes the decision rule sets of both interactions, which explains the SEQ_d routing type of the "Deliver PCs"-interaction in the global output diagram (see Figure 12).

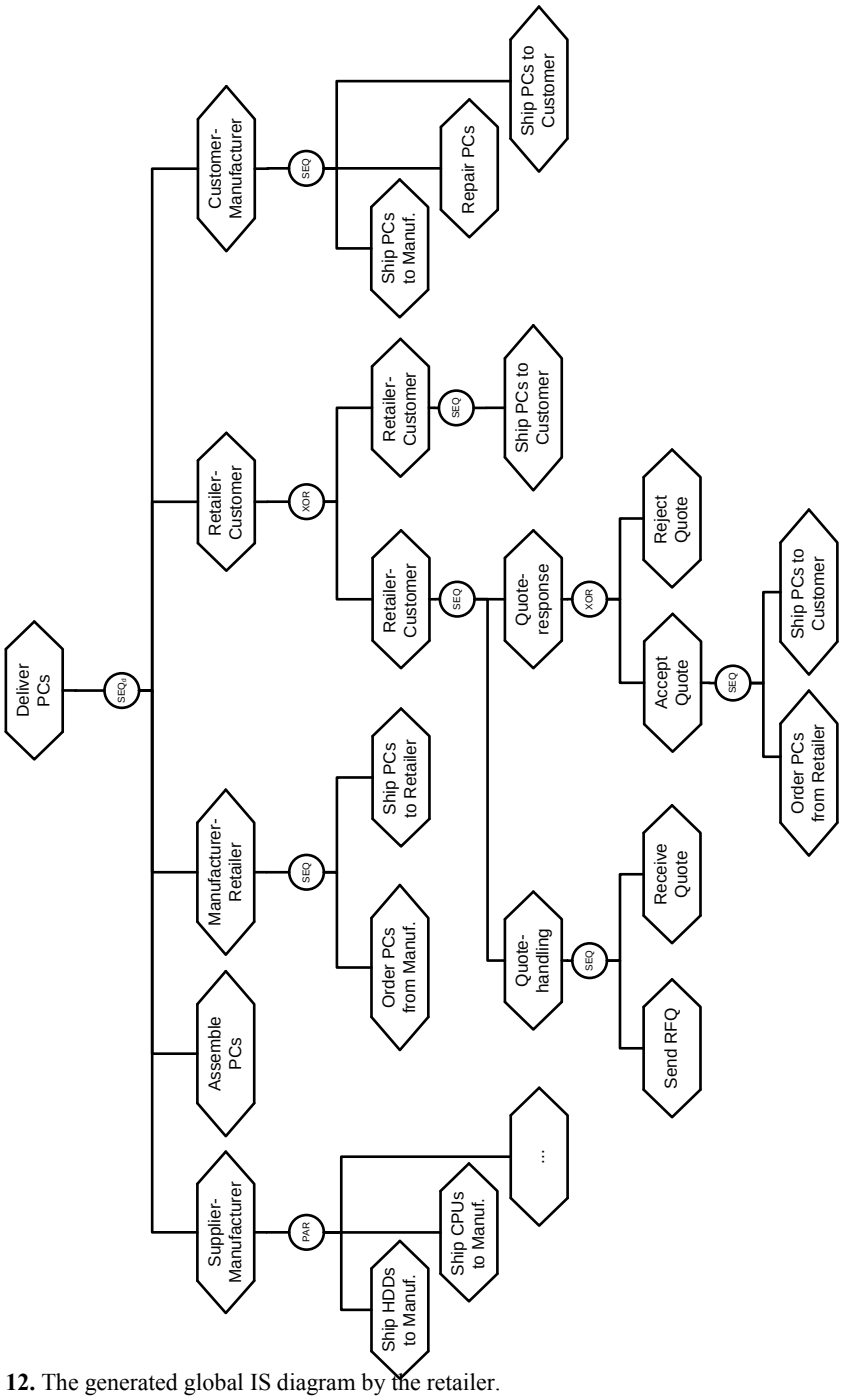

Fig. 12. The generated global IS diagram by the retailer.

6 Public and Private Process Representations

In a business domain where no product-centric data is available, the global construction algorithm can be used by several partners to build a shared global interaction model that coordinates (and possibly enacts) the overall collaborative business process. Each partner can represent its own local interactions that are part of the collaborative business process in an IS diagram. All local IS diagrams can then be used as input for the algorithm. In such a business setting, one of the issues is how much each organization (with its own legal and business boundaries) is willing to share with its partners. In the design of inter-organizational workflows, 'local' modelers strive to keep details opaque as much as possible. This attitude is based on the argument that organizations in general tend to keep their internal workings (i.e. process models) hidden. Though, a certain degree of disclosure is inevitable, especially in collaborative domains, in order to achieve good cross-organizational business process alignment and interoperability.

In TALL, a separation between public and private process views can be made on the same lines as the separation between interaction descriptions and agent behavior descriptions as discussed in Section 2. The agents are involved in interactions that have shared names, routing types, decision rules and roles, and exhibit behaviors that show their internal workings. All partners (and their internal agents) can hide their behaviors and expose only their local interaction structures (i.e. local IS diagrams) to their partners as input for the global construction algorithm. In this approach, the agent behaviors constitute private business process descriptions. Interoperability is achieved by the global IS diagram that is built, using the global construction algorithm, from the local IS diagrams.

This separation between public and private models only works if the private process descriptions of all the participants are aligned. Otherwise, a minimal disclosure of the behavioral touch points (i.e. the messages send and received) and their sequence is necessary in order to achieve simple alignment [22]. One agent has to inform the other agent what and when it is expected to send and receive a message during an interaction. Still, this level of disclosure does not reveal the exact activities in the private descriptions since only the message exchange points are made public. In conclusion, the following two views on public and private process representations apply to TALL:

- if the goal is to have minimal process disclosure, then only the local IS diagrams are made public for coordination of the overall collaborative business process using a shared global IS diagram. The agent behaviors are kept completely private;
- if the goal is to have more alignment, the agent behaviors can be made partially public by sharing the behavioral touch points.

Related work, like [1], separates between private and public process levels and recognizes the need for an organization to hide parts of its private process implementation. However, it is also mentioned that too much hiding can be counter-productive. In [19], the need for a process abstraction concept is stressed in the design and analysis of collaborative business processes. Such a concept must be able to represent internal or private processes and at the same time provide a process-oriented interface to the outside world, facilitating interweaving into partner processes. The merging of local IS diagrams, using the global construction algorithm, into a global IS

diagram provides such a feature. Other works show how public representations can be (formally) translated from internal representations [13]. Moreover, modeling languages like WS-BPEL [25] and BPMN [37] distinguish between public and private process representations. However, the public (named also abstract) process is a mere non-deterministic protocol that describes possible message exchanges. TALL brings an interaction structure that can and should be made public, and has an option to decide on the degree of exposure of the agent behaviors.

7 Discussion and Future Work

All agents involved in a collaborative business process have their own local, distributed views. Typically, no agent will have a complete view of all the interactions that occur in the process. In other words, each agent owns one or more local IS diagrams. Since interactions may influence each other, it makes sense to build an IS diagram that shows a global picture of the entire process. This global IS diagram increases process visibility and the understandability of the agent with regard to the supply chain. In the presented supply chain scenario, this is the retailer agent. The global IS diagram provides a more complete description of the entire supply chain, that is, it serves as a presentation of the multiple interactions that occur in the supply chain.

Such global models are especially useful when a set of parties interact in such a way that none of them sees all messages being exchanged, yet interactions taking place between some parties have an impact on the way other parties interact [39]. The presented approach only reveals the interactions with and between partners based on the available product-centric data. Partners do not have to share their behaviors, that is, their private business processes. In this way, interoperability can be achieved in situations where the (complete) models of behavior are not shared. Moreover, the global IS diagram could be used for analysis purposes. This potentially enables the retailer agent to better align its processes with the processes of its partner agents.

The presented example of product-centric data for the supply chain scenario contains data about one product individual. Future research intends to investigate how interaction diagrams can be build using product-centric data that contains data about multiple products or components with slightly different lifecycles.

The global construction algorithm can be improved by enhancing the **CONFLICT** function. First, an enhanced version of the **CONFLICT** function should, in some cases, return FALSE even if the interactions have different routing types in the global and local IS diagram. For instance, the PAR routing type is quite weak in the sense that it allows interactions to be executed in parallel or in any order. Therefore, it could be possible to merge two interactions with similar labels but with routing types PAR and SEQ. Second, the function should, in some cases, be able to merge alternative views with the same interactions occurring on different heights in the tree. In the current version of the **CONFLICT** function, the “Retailer-Customer”-interactions in the retailer and customer diagram are considered alternative views because the “Ship PCs to Customer” interaction appears on a different height in the retailer diagram. It is for instance possible that the “Ship PCs to Customer”-interaction in the customer

diagram is a secondary shipment. Thus, in the output the “Ship PCs to Customer”-interaction appears twice in the subtree of the highest level “Retailer-Customer”-interaction. Although it is possible that both “Ship PCs to Customer”-interactions are the same (i.e. the customer diagram may be based on incomplete product-centric data), it is considered more important to represent the views of all the agents even though this can lead to redundant interactions in the output. In line with the goal to create a minimal interaction representation, future research will investigate an improved version of the algorithm that reduces the number of (redundant) interactions in the output. Based on this, a third possible enhancement to the algorithm would be a “pruning”-feature, which compresses the global output diagram, by removing redundant interactions. Besides possible redundant views like the “Ship PCs to Customer”-interaction above, interactions can also be redundant when they have the same routing type as their parents. Often, the children of a redundant interaction can be directly connected to the parent of the redundant interaction. Together with these improvements, a mathematical evaluation of the algorithm in terms of formally specified properties like termination, correctness, complexity etc. is planned for the future.

As mentioned in Section 6, the global construction algorithm can also be used in cross-organizational environments where no product-centric data is available. In this case, the global construction algorithm can be used to build a shared global interaction diagram that manages the (distributed) behaviors of all the partners. Such a global interaction structure could be implemented on a collaborative IT platform and act as a coordination framework. This line of research is to be explored in more detail.

8 Conclusion

In product-centric supply chains, coordination and control efforts are aimed at individual products instead of orders, production orders or shipment batches like in conventional supply chains. The TraSer project is concerned with research into product-centric supply chains. In the TraSer approach, product-centric data is stored in a database that is associated to each product individual. Each product contains a label in the form of an RFID or barcode, which contains an ID@URI code. Besides the unique product identification number, this code contains an URI (Uniform Resource Identifier) that refers to the location of the associated database. Product visibility is achieved in the supply chain when each partner can access the available product-centric data.

This paper uses an illustrative example to show how a partner can use product-centric data to build a set of local interaction diagrams in which each diagram models the perspective of one partner in the supply chain. The interaction diagrams are part of a novel graphical modeling language named TALL, developed and adopted by the authors, which models collaborative business processes as a structure of role-based interactions in which agents play the roles. A formal method is introduced to enable the partner to automatically construct a global interaction diagram from the set of local interaction diagrams. In this way, the partner obtains a global view of the supply chain. The formal method comprises an algorithm that processes and compares all

local diagrams, detects any conflicts, and outputs a global interaction diagram. The global interaction diagram depicts a minimal representation where the interactions from the local diagrams are merged when there is no conflict.

By representing all views of the agents involved, the global interaction diagram increases process visibility and understandability in the supply chain for the partners that use the presented approach. This benefits process alignment and enterprise operability in the supply chain.

References

1. van der Aalst, W.M.P., Weske, M.: The P2P Approach to Interorganizational Workflows. In Dittrich, K.R., Geppert, A., Norrie, M.C. (Eds) CAiSE'01. LNCS, vol. 2068, pp. 140--156. Springer, Heidelberg (2001)
2. Bauer, B., Müller, J.P., Odell, J.: Agent UML: A Formalism for Specifying Multiagent Software Systems. The Int. J. of Software Engineering and Knowledge Engineering. 11(3), 207--230 (2001)
3. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, L.: Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems. 8(3), 203--236 (2004)
4. Bussmann, S., Schild, K.: Self-organizing Manufacturing Control: An Industrial Application of Agent Technology. In: 4th International Conference on Multi-Agent Systems, pp. 87--94. IEEE Computer Society, Washington, DC, USA (2000)
5. Caire, G., Coulier, W., Garijo F., Gomez, J., Pavon, J., Leal. F., Chainho, P., Kearney, P., Stark, J., Evans, R., Massonet, P.: Agent Oriented Analysis using Message/UML. In: Wooldridge, M.J., Weiss, G., Ciancarini, P. (Eds.) AOSE 2001. LNCS, vol. 2222, pp. 119--135. Springer, Heidelberg (2002)
6. Callon, M., Méadel, C., Rabeharisoa, V.: The Economy of Qualities. Economy and Society. 31(2), 194--217 (2002)
7. Camarinha-Matos, L.M., Afsarmanesh, H.: Collaborative Networks: A New Scientific Discipline. Journal of Intelligent Manufacturing. 16(4/5), 439--452 (2005)
8. Cervenka, R., Trenanský, I.: The Agent Modeling Language – AML: A Comprehensive Approach to Modeling Multi-Agent Systems. Birkhäuser, Basel (2007)
9. Collins, J., Aruachalam, R., Sadeh N., Eriksson, J., Finne N., Janson, S.: The Supply Chain Management Game for the 2007 Trading Agent Competition. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA (2006)
10. DeLoach, S.A., Wood, M.F., Sparkman, C.H.: Multiagent Systems Engineering. The Int. J. of Software Engineering and Knowledge Engineering. 11(3), 231--258 (2001)
11. Desel, J.: Process Modeling using Petri Nets. In: Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M. (Eds). Book Chapter. Process-Aware Information Systems: Bridging People and Software through Process Technology, pp. 147--177. John Wiley & Sons, Hoboken, New Jersey (2005)
12. Dumas, M., ter Hofstede, A.H.M.: UML Activity Diagrams as a Workflow Specification Language. In: Martin, G., Kobryn, C. (Eds.) UML 2001. LNCS, vol. 2185, pp. 76--90. Springer-Heidelberg (2001)
13. Eshuis, R., Grefen, P.: Constructing Customized Process Views. Data & Knowledge Engineering. 64(2), 419--438 (2008)
14. Harrison-Broninski, K.: Human Interactions: The Heart and Soul of Business Process Management. Meghan-Kiffer Press, Tampa, Florida, USA (2005)

15. Holmström, J., Främling, K.: Product Centric Integration: Exploring the Impact of RFID and Agent Technology on Supply Chain Management. In: Camarinha-Matos, L.M., Afsarmanesh, H., Ollus, M. (Eds.) PRO-VE'06. IFIP, vol. 224, pp. 565--572. Springer, Heidelberg (2006)
16. Holmström, J., Kajosaari, R., Främling, K., Langius, E.: Roadmap to Tracking Based Business and Intelligent Products. Computers in Industry. 60(3), 229--233 (2009)
17. Huvio, E., Grönvall, J., Främling, K.: Tracking and Tracing Parcels using a Distributed Computing Approach. In: Solen, O. (Eds.), 14th Annual Conference for Nordic Researchers in Logistics, pp. 12--14. Norwegian University of Science and Technology, Trondheim, Norway (2002)
18. Jagdev, H.S., Thoben, K.D.: Anatomy of Enterprise Collaboration. Production Planning & Control. 12(5), 437--451 (2001)
19. Lippe, S., Greiner, U., Barros, A.: A Survey on State of the Art to Facilitate Modelling of Cross-Organisational Business Processes. In Nüttgens, M., Mendling, J. (Eds.) 2nd German Informatics Society Workshop on XML Interchange Formats for Business Process Management, pp. 7--22. Gesellschaft für Informatik, Karlsruhe, Germany (2005)
20. Mayer, R.J., Painter, M.K., deWitte P.S.: IDEF Family of Methods for Concurrent Engineering and Business Reengineering Applications. Technical Report, Knowledge Based Systems, Inc., http://www.idef.com/pdf/IDEFFAMI.pdf (1992)
21. Melao, N., Pidd, M.: A Conceptual Framework for Understanding Business Processes and Business Process Modelling. Information Systems Journal. 10(2), 105--129 (2000)
22. Meyer, G.G., Szirbik, N.B.: Anticipatory Alignment Mechanisms for Behavioral Learning in Multi Agent Systems. In: Butz, M.V., Sigaud, O., Pezzulo, G., Baldassarre, G. (Eds.) ABiALS'06. LNCS, vol. 4520, pp. 325--344. Springer, Heidelberg (2007)
23. Meyer, G.G., Szirbik, N.B.: Agent Behavior Alignment: A Mechanism to Overcome Problems in Agent Interactions During Runtime. In: Klusch, M., Hindriks, K., Papazoglou, M.P., Sterling, L. (Eds.) CIA 2007. LNCS, vol. 4676, pp. 270--284. Springer, Heidelberg (2007)
24. Meyer, G.G., Främling, K., Holmström, J.: Intelligent Products: A Survey. Computers in Industry. 60(3), 137--148 (2009)
25. Organization for the Advancement of Structured Information Standards (OASIS): Web Services Business Process Execution Language Version 2.0. OASIS Standard, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf (2007)
26. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley & Sons, Chichester, West Sussex, England (2004)
27. Pesic, M., van der Aalst, W.M.P.: A Declarative Approach for Flexible Business Process Management. In: Eder, J., Dustdar, S. (Eds.) BPM 2006. LNCS, vol. 4103, pp. 169--180. Springer-Heidelberg (2006)
28. Presley, A.R., Liles, D.H.: A Holon-Based Process Modeling Methodology. Int. J. of Operations & Production Management. 21(5/6), 565--581 (2001)
29. Roest, G.B., Szirbik, N.B.: Escape and Intervention in Multi-Agent Systems. AI & Society. 24(1), 25--34 (2009)
30. Scheer, A.W., Thomas, O., Adam, O.: Process Modeling using Event-Driven Process Chains. In: Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M. (Eds). Book Chapter. Process-Aware Information Systems: Bridging People and Software through Process Technology, pp. 119--145. John Wiley & Sons, Hoboken, New Jersey (2005)
31. Stuit, M., Szirbik, N.B., de Snoo, C.: Interaction Beliefs: A Way to Understand Emergent Organizational Behaviour. In: 9th International Conference on Enterprise Information Systems, pp. 241--248. INSTICC, Funchal, Madeira, Portugal (2007)
32. Stuit, M., Szirbik, N.B.: Modelling and Executing Complex and Dynamic Business Processes by Reification of Agent Interactions. In: O'Hare, G., Ricci, A., O'Grady, M.,

Dikenelli, O. (Eds.) ESAW'06. LNCS, vol. 4457, pp. 106--125. Springer, Heidelberg (2007)
33. Stuit, M., Szirbik N.B.: Towards Agent-based Modelling and Verification of Collaborative Business Processes. Int. J. of Cooperative Information Systems. Under Review (2009)
34. Sturm, A., Dori, D., Shehory, O.: Single-Model Method for Specifying Multi-Agent Systems. In: 2nd International Conference on Autonomous Agents and Multiagent Systems, pp. 121--128. ACM, New York (2003)
35. Taveter, K., Wagner, G.: A Multi-Perspective Methodology for Modelling Inter-Enterprise Business Processes. In: Arisawa, H., Kambayashi, Y. (Eds.) ER'01. LNCS, vol. 2465, pp. 403--416. Springer, Heidelberg (2002)
36. Wagner, G.: The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behaviour. Information Systems. 28(5), 475--504 (2003)
37. White, S.A.: Introduction to BPMN. BPTrends White Papers & Technical Briefs, http://www.bptrends.com/deliver_file.cfm?fileType=publication&fileName=07%2D04%20WP%20Intro%20to%20BPMN%20%2D%20White%2Epdf (2004)
38. Wooldridge, M., Jennings, N.R., Kinny, D.: The GAIA Methodology for Agent-Oriented Analysis and Design. Autonomous Agents and Multi-Agent Systems. 3(3), 285--312 (2000)
39. Zaha, J.M., Dumas, M., ter Hofstede, A., Barros, A., Decker, G.: Service Interaction Modeling: Bridging Global and Local Views. In: 10th IEEE International Enterprise Distributed Object Computing Conference, pp. 45--55. IEEE Computer Society, Washington, DC, USA (2006)