Chapter 3

An algorithm to integrate local interaction diagrams in human collaboration processes

This chapter is an edited version of the manuscript:

Marco Stuit and Gerben G. Meyer.
Agent interaction modelling based on product-centric data: A formal method to improve enterprise interoperability.
Abstract

This chapter shows how organizations can use product-centric data to produce local interaction representations of the interactions with and between their partners. An interaction-centric business process modelling language named TALL, which is both graphical and formal, provides notations and diagrams to create these representations. A novel algorithm is introduced that enables partners to automatically integrate multiple local interaction diagrams into a global interaction diagram. This global interaction diagram improves enterprise interoperability since it increases overall process visibility. A process example of a simplified collaborative supply chain is used to exemplify the algorithm.
3.1. Introduction

Nowadays, there is an ongoing trend towards the provision of business processes that are collaborative in nature. Such business processes are provided by autonomous partners, which together deliver complex products and/or services in a collaborative network (Camarinha-Matos & Afsarmanesh, 2005; Moitra & Ganesh, 2005; Zacarias, Pinto, Magelhães, & Tribolet, 2009). This is explicitly shown by the emergence of new organizational forms like the networked or virtual enterprise (Jagdev & Thoben, 2001). The distributed nature of collaborative business settings is a good match for agent technology (Presley & Liles, 2001; Taveter & Wagner, 2002). The autonomous partners in such business settings 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 (Stuit & Szirbik, 2007; Stuit & Szirbik, 2009; Stuit & Wortmann, 2010). TALL is an interaction-centric business process modeling language, based on agent-oriented concepts and notations, to model the process structure of interactions that makes up a human collaboration process (HCP). The algorithm presented in this chapter adopts the language to create graphical interaction 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 (Bussmann & Schild, 2000). Supply chains are evolving 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 (Holmström & Främling, 2006), as shown in Figure 3-1. A conventional supply chain is based on a systems design that is focused on location-specific material accounts and transactions between locations. In contrast, a product-centric supply chain is based on a systems design that tracks and controls individual products independently of the location and the ownership of the product individual. Callon, Méadel, and Rabeharisoa (2002) explain that product-centric control makes explicit the role of the product as the coordinating entity in the delivery of customized products and services.
Product-centric control is the focus of the TraSer project\textsuperscript{20}, in which control efforts are directed at individual products of the value-adding (supply) network. The idea behind product-centric control is that individual products and their components are the basic entities in the information system rather than orders, production orders, or shipment batches. In the TraSer approach, each product is labelled with an RFID or barcode, which contains an ID@URI code (Huvio, Grönvall, & Främling, 2002). With this ID@URI code, each product carries a unique identification number (ID), as well as a reference (Uniform Resource Identifier) to the location of a database where 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.

Product-centric data is information that is linked to individual products to meet information management needs of partners and/or customers at the level of the individual product (and its components). Such needs usually arise in supply networks for tracking and logistics control (Kärkkäinen, Ala-Risku, & Främling, 2003). To collect and share product-centric data, peers can directly interact with an information-based representation of a product. This information-based representation acts like an interface to a company-specific database that stores and maintains the product-centric data. The connection between the physical product and the information-based representation is usually made using a tag (e.g. RFID tag) and a reader (e.g. a handheld RFID reader) (Meyer, Främling, & Holmström, 2009).

\textsuperscript{20} \url{http://www.traser-project.eu}
The product visibility that results from the use of product-centric control in the network does not imply process visibility. This chapter proposes to use product-centric data to identify the multiple interactions between the different partners in a supply chain. Together these interactions form a HCP. Currently, none of the partners has a global view of the entire HCP in the supply chain, as each partner only has data about their interactions with direct partners. To increase interoperability between the partners, it is considered beneficial to build a global interaction diagram, which gives and end-to-end overview of the HCP. By using the product-centric data available in the supply chain, each partner is enabled to identify and model the local interactions of partners in local interaction diagrams. This chapter presents a novel algorithm, named the Global Construction Algorithm (GCA), which enables partners to automatically integrate the local interaction diagrams into a global interaction diagram. This global interaction diagram realizes improved understanding and insight in the supply chain’s HCP for a given partner. The goal of GCA is to build a minimal global interaction diagram where interactions from different local interaction diagrams are merged when there is no conflict. A conflict indicates the existence of alternative local views on the same interaction. GCA assumes a common shared ontology in the business domain under consideration. Ontology matching is outside the scope of this chapter.

This chapter is structured as follows. Section 3.2 introduces the notations of the TALL interaction diagrams. Next, Section 3.3 shows an example HCP, concerned with a simplified supply chain, in which product-centric data is used to build local interaction diagrams. After, Section 3.4 presents GCA. Section 3.5 applies GCA to the case example. After, Section 3.6 presents a discussion together with future work. Finally, Section 3.7 gives conclusions.

3.2. The TALL modelling language

TALL is an interaction-centric business process modelling language inspired by the agent paradigm (Stuit & Szirbik, 2007; Stuit & Szirbik, 2009; Stuit & Wortmann, 2010). The language is both formal and graphical. The organizational context in which a HCP occurs is seen as a multi-agent environment in which different agents behave in interactions to coordinate their work. The language explicitly recognizes interaction as the core activity in collaborative organizational work.

The TALL Interaction Structure (IS) diagram represents the interactions in a given HCP. In the IS diagram, interactions are represented by flattened hexagons, and are related to other interactions through composition (one interaction being
part of another) and routing (one interaction must be completed before, in parallel with, or instead of another interaction). Figure 3-2 shows an example of an IS diagram in which interaction X is decomposed in two child interactions. Formally, the IS diagram is a tree with a single root interaction. This implies that each interaction appears at a certain level or height in the tree. The root interaction, which appears at the top, has height zero. Roles, which are depicted as ellipses, are connected to the lines outgoing the hexagon (see Figure 3-2). In TALL, an interaction does not exist without at least two roles being bound to it. For simplicity, a role can be thought of as a generic participant type rather than a specific agent who plays a role in an interaction. Roles also represent division of responsibility in the process. Roles are specified for each elementary interaction (i.e. interactions without children) and then collected bottom-up as in Figure 3-2.

Figure 3-2. Agent interaction modelling in the TALL modelling language.

The routing relations supported in the IS diagram are sequential routing (SEQ), parallel routing (PAR), and exclusive choice (XOR). Child interactions are linked to their parent interaction by a routing symbol that indicates the routing relation between a given set of sibling interactions. As Figure 3-2 shows, the routing symbol is a circle, which graphically appears below parent interactions. With sequential routing, there is a strict dependency between the interactions (i.e. an ordered sequence). With parallel routing, the interactions can be performed at the same time (i.e. true concurrency) or in an arbitrary order (i.e. an unordered sequence). Decision rules can be attached to the routing types to enable rule-based execution of interactions. Graphically, a (non-empty) decision rule set is depicted by attaching a subscripted letter \(d\) to the routing type (as in Figure 3-2). Decision rules 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\(_d\)). Graphically, for the SEQ and PAR routing types (with or without decision rules), sibling interactions are laid out linearly in the order of execution with the first interaction on the left. In Figure 3-2, interactions X-1 and X-2 are performed in sequence to complete interaction X. Interaction X-2 is completed by either interactions X-3 or X-2. In general, a parent interaction completes when its child interactions complete according to the specified routing relation. Thus, by convention, completion or composition is read bottom-up while routing between sibling interactions is read from left to right. In this chapter, the IS diagram is used to represent the interactions between agents in a supply chain.

A visual editor is available for the TALL modelling language to build and manipulate TALL diagrams. Moreover, a software tool, named Local IS Integrator (LISI), which works on the database created by the visual editor implements GCA. All software tools are freely available from the software section on http://www.agentlab.nl/.

### 3.3. Product-Centric Case Example

This section presents an example HCP from the perspective of one agent in a supply chain. The example HCP is used to illustrate how this one agent can build a global IS diagram 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 example HCP is a simplified supply chain, based on (Collins, Aruachalam, Sadeh, Eriksson, Finne, & Janson, 2006), in which agents have the shared goal of delivering Personal Computers (PCs) to end customers. Figure 3-3 shows a high-level picture of the example HCP.

![Figure 3-3. The simplified collaborative supply chain.](image)
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. Here, the focus is on the view of the retailer agent in the example HCP. The retailer can build a (local) IS diagram of its interactions with its direct partner agents. This diagram is shown in Figure 3-4. The roles are omitted in this diagram for clarity reasons. However, the agents are playing their given 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 entire supply chain. To achieve this goal (i.e. to complete the Deliver PCs interaction), the retailer interacts with the manufacturer (interaction Manufacturer-Retailer) and with the customer (interaction Retailer-Customer). With regard to the Manufacturer-Retailer interaction, the retailer agent interacts with the manufacturer 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 by the customer, a quote is sent to the customer, and if the customer accepts the quote, an order and shipment of the PCs follows.

Figure 3-4. The IS diagram of the retailer.
To improve the insight of the retailer agent in the example HCP, the retailer agent can build local IS diagrams from the perspective of the other agents in the supply chain by generalizing over available product-centric data. Table 3-1 shows an example of product-centric data of an individual product in the supply chain. The product-centric data provides partial information about the interactions in which the other agents are involved, as the product-centric data per product individual only contains information about product location and shape changes including the time when these changes happened. Based on this, the local IS diagrams that are input to GCA are assumed to be incomplete.

Table 3-1. Example of product-centric data.

<table>
<thead>
<tr>
<th>Product-ID</th>
<th>Time</th>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>10.01.2008 12:34</td>
<td>Shipment</td>
<td>Manufacturer X</td>
</tr>
<tr>
<td>123456</td>
<td>15.01.2008 13:35</td>
<td>Assemble</td>
<td>Manufacturer X</td>
</tr>
<tr>
<td>123456</td>
<td>20.01.2008 14:36</td>
<td>Shipment</td>
<td>Retailer Y</td>
</tr>
<tr>
<td>123456</td>
<td>25.01.2008 15:37</td>
<td>Shipment</td>
<td>Customer Z</td>
</tr>
</tbody>
</table>

From the product-centric data only the interactions with and between partners can be identified. The activities executed by the partners to perform the interactions remain private knowledge. Furthermore, confidentiality is provided to partners as they can decide to share or withhold certain product-centric data. The number of interactions identified from the product-centric data is higher in business settings where trust between partners is high. In such settings, 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 from the perspective of the manufacturer agent can be built by the retailer agent, based on the available product-centric data. Figure 3-5 illustrates this diagram for the example HCP. 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 modelled as an interaction, as it can be considered an interaction between the manufacturer and the product. An example is the interaction Assemble PCs in Figure 3-5.
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 have a single child, since the interactions that are identified based on the product-centric data can be incomplete like for instance the Manufacturer-Retailer interaction in Figure 3-5. The Customer-Manufacturer interaction does only occur when one or more of the delivered PCs need to be repaired by the manufacturer agent. Thus, a decision rule that can check this condition is attached to the SEQ routing type that graphically appears below the Deliver PCs interaction (see Figure 3-5). If one or more of the PCs need to be repaired, all four interactions on level one in the tree occur in sequence. If not, N-out-of-M children occur in sequence. In this specific example, the latter case means that three out of four children occur in sequence.

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 3-6. Again, based on the product-centric data, only the shipments of and physical changes to the product are revealed.
3.4. Global Construction Algorithm

This section introduces the formal aspects of GCA, which can integrate a set of local IS diagrams into a global IS diagram. As explained in the previous section, each partner agent involved in a product-centric supply chain 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 (e.g. in a supply chain) are not always willing to exchange their process definitions.

Figure 3-7 shows the input and output of GCA 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 HCP under study. Furthermore, GCA expects a set of local IS diagrams $LI$ (Local Input) that are to be merged. Each local IS diagram is based on the same root interaction $i_0$. GCA outputs a global diagram $GO$ (Global Output). As explained before, the goal of the algorithm is to build a minimal interaction diagram $GO$ 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 below).
GCA uses some notations from the formal definition of the TALL IS diagram (Stuit & Szirbik, 2009). Here, these notations are explained informally:

- $I$ is the set of interactions that forms 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 a set of sibling interactions is routed to complete their parent interaction. As Section 3.2 explains, the supported routing types are SEQ, PAR, and XOR with or without decision rules;
- The partial ordering relation $<, \subseteq I \times I$ 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 interaction (as in Figure 3-2);
- $DR(i)$ is a function that assigns a decision rule set to each interaction. As Section 3.2 explains, 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 connects roles to interactions. Graphically, this is depicted by a connector line between a role and an interaction (as in Figure 3-2);
- $L_R(r)$ 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 $i$. 

Figure 3-7. Illustration of the input and output of the global construction algorithm.
In addition, some new operators and notations are used in GCA, as an extension of the formal definition in (Stuit & Szirbik, 2009):

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 $IS_{IS_1 \cup IS_2} = IS_1 \cup IS_2$ such that: $I_{IS_1 \cup IS_2} = I_{IS_1} \cup I_{IS_2}$, $<I_{IS_1 \cup IS_2} = <I_{IS_1} \cup <I_{IS_2}$, $R_{IS_1 \cup IS_2} = R_{IS_1} \cup R_{IS_2}$, $RI_{IS_1 \cup IS_2} = RI_{IS_1} \cup RI_{IS_2}$. 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 $I_{I_1 \cup I_2} = I_1 \cup I_2$ such that: $L_{I_{I_1 \cup I_2}} = L_{I_{I_1}} \cup L_{I_2}$, $RT_{I_{I_1 \cup I_2}} = RT_{I_{I_1}} \cup RT_{I_2}$, $DR_{I_{I_1 \cup I_2}} = DR_{I_{I_1}} \cup DR_{I_2}$. 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 $R_{R_1 \cup R_2} = R_1 \cup R_2$ such that: $L_{R_{R_1 \cup R_2}} = L_{R_1} \cup L_{R_2}$;

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 3-8);

3. $comp(i)$ is a function that returns a single component of the IS diagram. In other words, it returns a subtree with the single interaction $i$. 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 by GCA to differentiate between ‘normal’ interactions and dummy interactions. Dummy interactions are introduced during execution of GCA to act as the parent of identically labelled interactions, which the algorithm cannot merge because of a conflict. Such interactions are alternative views on the same interaction. This implies that all children of a dummy interaction have identical labels. Initially, the default setting for all interactions is $dum(i) = FALSE$: $\forall i \in I: \neg dum(i)$;

6. $IH$ is an ordered set in which all interactions $i \in I$ are ordered according to the binary relation $R = \{(i_1, i_2) \mid i_1 \in I \land i_2 \in I \land ht(i_1) \leq ht(i_2)\}$. Effectively, this means that the interactions in the set $IH$ are ordered according to their height in an ascending order;

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.
GCA is listed in Table 3-2. The algorithm is expressed in pseudo-code, which uses set-theoretic notation mixed with the control structures of conventional high-level programming languages.
Table 3-2. The 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))\GO then
7:         \li_new := clone(i)\li
8:         GO := GO ∪ \{comp(\li_new)\}
9:       \li_i := \li_i \cup \{(parent(i))\GO, \li_new\}
10:      else if conflict(subtree(i)\GO, subtree(i)\li) then
11:         \li_dummy := j: j ∈ \i_G \land i = j \land dum(j)
12:        if \li_dummy \land \exists then
13:           \li_dummy := clone(i)\GO
14:           \DOM(\li_dummy) := ∅
15:           dum(\li_dummy) := TRUE
16:           GO := GO ∪ \{comp(\li_dummy)\}
17:           \li_i := \li_i \cup \{(parent(i))\GO, \li_dummy\}
18:           \li_i := \li_i \setminus \{(parent(i))\GO, \li_double\}
19:           \li_i := \li_i \cup \{\{\li_dummy, \li_double\}\}
20:       \li_conflict := clone(i)\li
21:       \li_conflict \setminus \{\li_double\} := \li_conflict \setminus \{\li_double\}
22:       \RI(\li_dummy) := XOR
23:       GO := GO ∪ \{comp(\li_conflict)\}
24:       \li_i := \li_i \cup \{\li_dummy, \li_conflict\}
25:      else
26:        \li_i := \li_i \cup \{\li_conflict\}
27:       return GO
28:   end

GCA is contained into a main for each-loop (Lines 4-27) that in each iteration compares GO (= GI, see Line 3 in Table 3-2) to a local IS diagram li ∈ LI. During the comparison, the algorithm searches for commonalities (i.e. overlapping views) and/or differences (i.e. alternative or complementary views) between GO and li by comparing (labels of) the interactions in both diagrams. If possible, GCA merges interactions in both diagrams. In any case, GCA produces a new (integrated) global IS diagram GO 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, GCA produces a global IS diagram GO in which all local IS diagrams are integrated.

GCA first assigns GI to GO (Line 3). Inside the main for each-loop (Lines 4-27), an inner for each loop (Lines 5-26) processes one by one all the interactions
from the local IS diagram \( li \) under consideration according to the ordering in the set \( IH \). This means that GCA starts with the root interaction in the local IS 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 on until all the interactions in the local IS diagram \( li \) have been processed. 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 or subtree of \( GO \) (Line 6)**

The if statement in Line 6 of GCA tests whether interaction \( i^{li} \) should be added to \( GO \). Interaction \( i^{li} \) is added to \( GO \) when \( i^{li} \) does not exist in the subtree of its parent in \( GO \). When the condition in Line 6 is true (i.e. case one applies), a clone of \( i^{li} \) is created (Line 7), the clone is added to \( GO \) (Line 8), and the clone is connected to the correct parent in \( GO \) (Line 9).

Figure 3-9 shows a generic example (without roles) in which the condition in the if statement in Line 6 is true for interaction \( D^{li} \). In this specific example, interaction \( D^{li} \) does not exist in the subtree of its parent in \( GO \), that is, the subtree of \( A^{GO} \). Therefore, the component \( D^{li} \) is added to \( GO \). In Figure 3-9, \( GO_{old} \) and \( li \) are the two IS diagrams being compared. Both diagrams show the incomplete local interaction view of an agent, which is build based on product-centric data. In this regard, each IS diagram represents a specific incomplete part of the overall HCP in the product-centric supply chain. The local IS diagram of the first agent shows that interactions \( B \) and \( C \) occur in parallel to complete interaction \( A \). The local IS diagram of the second agent shows that interactions \( C \) and \( D \) occur in parallel to complete interaction \( A \). Based on the assumption that local IS diagrams are incomplete, it is sufficient to perform interaction \( A \) once in \( GO_{new} \). In this regard, the two agents provide complementary views with an overlap (i.e. interaction \( C \)) that are integrated to create a more complete interaction view. In the output diagram, GCA has merged the overlap. It is important to note that, merging only occurs when the views of agents are not conflicting. Case 2 discusses how GCA deals with a conflict situation.
Figure 3-9. 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 interaction $D^{li}$ is added to $GO_{new}$ since it does not exist in $GO_{old}$.

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

If the condition in Line 6 is false, which means interaction $i^{li}$ has a counterpart $i^{GO}$ in the same segment of GO, GCA continues with the else if statement in Line 10. This 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 3-3.
Table 3-3. The function CONFLICT.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>function CONFLICT (sGO, sli)</td>
</tr>
<tr>
<td>2</td>
<td>begin</td>
</tr>
<tr>
<td>3</td>
<td>if RT(l₀)GO ≠ RT(l₀)sli then</td>
</tr>
<tr>
<td>4</td>
<td>return TRUE</td>
</tr>
<tr>
<td>5</td>
<td>else</td>
</tr>
<tr>
<td>6</td>
<td>X := children(l₀)sli</td>
</tr>
<tr>
<td>7</td>
<td>Y := X ∩ GO</td>
</tr>
<tr>
<td>8</td>
<td>for each j ∈ Y do</td>
</tr>
<tr>
<td>9</td>
<td>if ht(j)GO ≠ ht(j)sli then</td>
</tr>
<tr>
<td>10</td>
<td>return TRUE</td>
</tr>
<tr>
<td>11</td>
<td>return FALSE</td>
</tr>
<tr>
<td>12</td>
<td>end</td>
</tr>
</tbody>
</table>

In general, a conflict means that the interactions being compared cannot be merged since they are alternative views on the same interaction. Hence, GCA keeps both interactions (iGO and ili) in the global output diagram. A new interaction idum (dummy interaction) is introduced (Line 13) to act as the parent of iGO and ili. Since idum carries the decision rule set of iGO (i.e. it is a clone of interaction iGO, see Line 13), Line 14 empties the decision rule set of idum. Next, Line 16 adds idum including all its attributes and roles to GO. GCA then assigns idum as parent interaction of iGO (Line 19). In the scenario in which iGO already has a parent, Lines 17 and 18 make sure that this parent is detached as parent of iGO and instead is assigned as parent of idum. 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 two interactions. Since idum acts as parent of iGO and ili, the next step is to add ili to GO. To this end, Line 20 first creates a clone of ili named icontlict. After, Line 23 adds icontlict to GO and Line 24 assigns icontlict as a child interaction of idum. Line 21 assigns the union of the components of iGO and icontlict to the dummy interaction idum. In this way, the combined roles of its child interactions are attached to idum. Since the direct child interactions of idum are different views on the same interaction, idum is assigned the routing type XOR by Line 22. The use of a different routing type than XOR would imply that the conflict interactions are executed more than once.

The function CONFLICT receives as input, from GCA, the subtrees of iGO and ili (i.e. sGO and sli, see Table 3-3). This means that the root interactions in the function CONFLICT are (the currently being processed) interaction ili and its counterpart iGO. A conflict arises in two situations. First, when the routing types of iGO and ili are different (Line 3 in Table 3-3), they are considered alternative views
on the same interaction. In this case, the **CONFLICT** function returns TRUE (Line 4 in Table 3-3). This is the first conflict situation. Second, when the routing types of $i^{GO}$ and $i^{li}$ are not different, the **CONFLICT** function builds a set $X$ that contains all child interactions of $i^{li}$ (Line 6 in Table 3-3), 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 3-3). For all the interactions in the set $Y$, the **CONFLICT** 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 $i^{li}$ have counterparts in $sGO$ that occur on another level. If there are no such counterparts, there is no conflict and the function returns FALSE (Line 11 in Table 3-3). However, if there are such counterparts (Line 9 in Table 3-3) then $i^{GO}$ and $i^{li}$ are considered alternative views on the same interaction and the function returns TRUE (Line 10 in Table 3-3). The latter forms the second conflict situation.

Figure 3-10 shows a generic example in which the interactions $A^{GO}$ and $A^{li}$ are in conflict because they have different routing types (i.e. the first conflict situation). A new (dummy) interaction with the same label and the routing type XOR is introduced in $GO_{new}$ to act as the parent of $A^{GO}$ and $A^{li}$.
Figure 3-10. A generic example of a Global Input diagram (GO\textsubscript{old}), a local input diagram (li), and a Global Output diagram (GO\textsubscript{new}). The example illustrates the case in which $A^\text{li}$ generates a conflict because a counterpart $A^\text{GO}$ exists with a different routing type.

Figure 3-11 depicts a generic example of the second conflict situation. In this specific example, interactions $A^\text{GO}$ and $A^\text{li}$ are in conflict because the parent interaction $A^\text{li}$ has a child $D^\text{li}$ with a counterpart $D^\text{GO}$ that exists in the subtree of $A^\text{GO}$ on a different level. Therefore, a dummy interaction $A$ with routing type $XOR$ is added to $GO_{\text{new}}$. 

102
An algorithm to integrate local interaction diagrams in human collaboration processes

Figure 3-11. A generic example of a Global Input diagram (GO\textsubscript{old}), a local input diagram (li), and a Global Output diagram (GO\textsubscript{new}). The example illustrates the case in which $A^{li}$ generates a conflict because it has a child interaction that occurs on a different level in the subtree of its counterpart $A^{GO}$. 

103
Case 3: Interaction $i^li$ exists in the same segment or subtree of $GO$, and there is no conflict situation (Line 25)

From here, any line numbers refer to Table 3-2 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 merged so that their roles and attributes are properly integrated. Figure 3-9 includes an example of this case. In Figure 3-9, parent interaction $A^li$ has a child interaction $C^di$ 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 $C^GO$ and $C^di$ occur on the same level in $GO_{old}$ and $li$. Thus, in this case, it is unnecessary to introduce a dummy interaction and keep both $A^GO$ and $A^li$ in the output diagram.

3.5. Application of the Global Construction Algorithm

This section applies GCA to the example HCP described in Section 3.3 and shows the output global IS diagram. The retailer agent in the supply chain can run GCA with the different local IS diagrams it has created based on the available product-centric data (as shown in Figure 3-4, Figure 3-5, and Figure 3-6) as input. Figure 3-12 shows the output diagram. This global IS diagram shows the example HCP starting from the supplier, until the product reaches the customer, and afterwards, when optionally a product (i.e. PC) is returned by the customer for repair by the manufacturer. Despite its simplicity, the example demonstrates several important features of GCA.

First, the example shows how complementary views on an interaction are integrated when a given interaction appears in the local diagram $li$ under consideration but does not appear in the global diagram $GO$. This reflects the first case in GCA. An example of such an interaction is the Assemble PCs interaction from the manufacturer diagram (see Figure 3-5), which does not appear in the retailer diagram (see Figure 3-4). A clone of this interaction is created, the clone is added to $GO$, and connected to the same parent in $GO$ (see Figure 3-12).
Figure 3-12. The output of the global construction algorithm for the retailer agent in the example HCP.
Second, the example shows how GCA deals with a conflict situation, that is, the second case in the algorithm. In the example, the retailer diagram (Figure 3-4) and the customer diagram (Figure 3-6) both include the Retailer-Customer interaction. The Ship PCs to Customer interaction, which is a child of the Retailer-Customer interaction in the customer diagram, also exists 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. GCA keeps both views in the output diagram, and adds a new (dummy) parent interaction for both views. In this way, the ordering constraints of both views are preserved. Since both views are alternative descriptions of the same interaction, the new parent interaction is assigned the XOR routing type (see Figure 3-12).

Third, the example shows how GCA deals with the third case. 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 level in the retailer diagram. Thus, the CONFLICT function returns FALSE and GCA executes the statement in Line 26 (see Table 3-2). This statement integrates the attributes and roles of both interactions. This includes the decision rule sets of both interactions, which explains the SEQd routing type of the Deliver PCs-interaction in the global output diagram.

3.6. Discussion
In a HCP, typically, no agent has a complete view of all the interactions that occur in the process. Each agent has its own local situated view of the process. This view can be made explicit in a local IS diagram. To obtain a global view of the HCP, the local IS diagrams can be integrated using GCA. The local IS diagrams can be build based on product-centric data like in the example HCP. Here, the output global IS diagram provides the retailer agent with an end-to-end overview of the interactions in the product-centric supply chain.

For other HCPs, which are performed in business settings where no product-centric data is available, the value of GCA is its ability to integrate a set of local IS diagrams, owned by different agents. More specifically, a modeller can capture and model the local (i.e. restricted and incomplete) interaction views of the process participants (read: agents) in local IS diagrams. These local IS diagrams are created separately (and concurrently when multiple modellers are involved) for each agent.
Next, GCA can be used to integrate the local IS diagrams into a global IS diagram. In this way, the tacit domain knowledge of agents can be formalized in local diagrams and be used to build a global diagram. Taking into account the local interaction views of the agents has the potential to minimize the gap between the actual HCP and the modelled HCP. The global IS diagram makes the interactions in the HCP under study explicit, and amenable to proper analysis and improvement.

In a process execution environment, the role of the global IS diagram could be to enact and/or monitor the execution of the interactions in a given HCP. The diagram could serve as a coordination structure for the activities executed by the distributed agents to perform the interactions, and thereby improve process interoperability and alignment. For this, the agents need to adopt a multi-agent system in which the global IS diagram can reside. The realization of such a multi-agent system is part of future research. An interesting research direction is the use of GCA at process execution time. During HCP execution, it should be possible for agents to initiate interactions that are not part of the IS diagram to ensure process flexibility in highly dynamic contexts. The algorithm can be used at runtime to make sure that local changes are reflected in the (global) IS diagram. In this way, on-the-fly structural modification of the IS diagram can be realized.

GCA can be improved by enhancing the CONFLICT function. First, an enhanced version of the CONFLICT function could, in some cases, return FALSE even if two interactions with identical labels have different routing types. For instance, the PAR routing type is quite weak in the sense that it allows interactions to be executed at the same time or in an arbitrary order. Therefore, it could be possible to merge two interactions with identical labels with routing types PAR and SEQ. Second, the function should, in some cases, be able to merge alternative views with identically labelled interactions occurring on different levels (i.e. the second conflict situation in GCA). Currently, the Retailer-Customer interactions in the retailer diagram (Figure 3-4) and the customer diagram (Figure 3-6) are considered alternative views because a descendant of these interactions (i.e. the Ship PCs to Customer interaction) appears on a different level in both diagrams. Although it is likely that both diagrams are concerned with the same Retailer-Customer interaction, it is also possible that for instance the shipment in the customer diagram refers to a secondary shipment. In this case, the Retailer-Customer interactions are different interactions. The direction chosen with GCA is to represent the views of all the agents so that no information is lost, even though
Chapter 3

this can lead to redundant interactions in the global output diagram. Therefore, the output diagram in Figure 3-12 shows both Retailer-Customer interactions. In line with the general goal of GCA to create a minimal interaction representation, future research intends to investigate ways to minimize redundancy in the output diagram. Related to this, a possible future enhancement to GCA would be a pruning feature, which compresses the global output diagram by removing redundant interactions. Besides redundancy that can result from alternative views, redundancy can also occur when an interaction has the same routing type as its parent interaction. Often, such an interaction can be deleted and its child interactions can be directly connected to its parent interaction.

3.7. Conclusions

In product-centric supply chains, coordination and control efforts are aimed at individual products. In these supply chains, product owners store product-centric data in a database that is associated to each product individual. When this database is made accessible to partners (e.g. via a product interface), product visibility is achieved in the supply chain.

This chapter presents an example HCP of the interactions between partners in a simplified product-centric supply chain. The purpose of the example is threefold. First, it shows how a partner can use product-centric data to identify its own interactions and the interactions of other partners in the supply chain. Second, it shows how to formalize the identified interactions in local interaction diagrams for each partner. Finally, it shows how a novel algorithm is used to automatically integrate the local interaction diagrams in a global interaction diagram.

The focus is on the exemplification of the formal components and workings of the algorithm. The goal of the algorithm is to build a minimal global interaction diagram where the interactions from the local interaction diagrams are merged when there is no conflict. A conflict indicates the existence of alternative local views on the same interaction. The interaction diagrams are visualized with an interaction-centric business process modelling language named TALL. The global interaction diagram increases process visibility, and benefits process alignment and operability in the HCP under study.