Towards Distributed Configuration

Alexander Felfernig, Gerhard E. Friedrich, Dietmar Jannach, and Markus Zanker

Institut f. Wirtschaftsinformatik und Anwendungssysteme
Universität Klagenfurt
email{felfernig, friedrich, jannach, zanker}@ifit.uni-klu.ac.at

Abstract. Shorter product cycles, lower prices, and the production of highly variant products tailored to the customer needs are the main reasons for the proceeding success of product configuration systems. However, today's product configuration systems are designed for solving local configuration tasks only, although the economic development towards webs of highly specialized solution providers demands for distributed problem solving functionality. In this paper we motivate the integration of several configurators and give a formal definition of the distributed configuration task based on a logic theory of configuration. Furthermore, we present a basic architecture comprising several configuration agents and propose an algorithm for cooperation between distributed configuration systems that ensures correctness and completeness of configuration results.

1 Introduction

Configurators are not only important enablers of the mass customization paradigm but also among the most successful applications of AI-technology. Configurators calculate product variants which fulfill customer requirements as well as technical and non-technical constraints on the product solution. As the digital economy of the 21^{st} century will be based on flexibly integrated webs of highly specialized solution providers, the joint configuration of organizationally and geographically distributed products and services must be supported. The rush for supply chain integration by web-based selling systems and electronic procurement offers new challenges for configuration technology. While supply chain integration of standardized, mostly well defined products can be quite well achieved, the case for complex configurable products and services is still an open research issue.
Current configuration technology [11] does not yet offer concepts and tools to support the integration of configuration systems. In particular, a distributed configuration problem cannot be solved by a single configurator with a centralized knowledge base for security and privacy reasons of suppliers. As we have to cope with distributedness, we must accept the fact that there is no central point of knowledge. Neither a main vendor nor any of its suppliers has full knowledge on the whole problem domain. The second point is heterogeneity; we cannot assume that each of the engaged parties, that have knowledge on a subset of the problem domain and can therefor provide to the overall solution, employs the same knowledge representation formalisms.
Consequently, our goal is to contribute to the further development of configuration technology such that distributed configuration problems can be solved. In order to give a

© Springer-Verlag Berlin Heidelberg 2001

clear and general problem definition, we base our contribution on a logical foundation. This abstract approach allows us to determine the basic requirements for the integration of special instances of configuration methods and tools found in research and industry. Our introductory example is based on an application scenario provided by one of our industry partners (Section 2). Based on the definition of a central configuration problem, we formally define the distributed configuration task (Section 3) and show under which conditions the central and distributed problems are equivalent. We present an algorithm which enables configuration agents to cooperatively construct valid configurations and describe the required properties in order to assure correctness and completeness (Section 4). Aspects of integration are adressed in Section 5. Finally we discuss related work followed by conclusions.

2 Motivating Example

We introduce our concepts by presenting a motivating scenario from the area of telecommunication systems. Our product example is a telecommunication switch for enterprise networks. The functionality of the switch can be extended by installing additional software modules onto the hardware component such as management software or application packages for messaging and ip-services. These additional applications are third-party products or may be developed by a subsidiary company. The customer, however, wishes to order a completely configured product solution, comprising the switching hardware and all needed add-ons. For obvious reasons each supplier maintains the product knowledge within its own sales configuration system that cooperates with others. In our scenario a facilitating agent coordinates the search for a configuration solution of three configuration agents representing the providers of the *switching hardware*, the *messaging* and the *ipvoice* application software add-ons. We employ a logic theory of configuration [2] that complies with the component-port representation for configuration knowledge [10]. This logical model serves as a general ontology for the configuration domain. We allow that all involved configurators may use a proprietary representation formalism. However it must be assured that the content of communicated messages can be mapped onto the concepts of this logical theory. In our example we use only $types$ to denote the set of component types while $ports$ describes their connection points. We do not employ attributes of types and their domains in this example, although these concepts are part of our logic theory of configuration.

$types = \{tecom, srack, lrack, ipvoice, swpack1, swpack2, msger, uppack\}.$
$ports(tecom) = \{rack, ipvoice, msger\}.$
$ports(lrack) = \{tecom\}. \; ports(srack) = \{tecom\}.$
$ports(ipvoice) = \{tecom, swpack\}.$
$ports(swpack1) = \{ipvoice\}.$
$ports(swpack2) = \{ipvoice\}.$
$ports(msger) = \{tecom, upgr\}.$
$ports(uppack) = \{msger\}.$

The predicates used for describing configurations are contained in a set $CONL$, where $CONL = \{type/2, conn/4\}$ for our example. A type t is associated with a component c by literal $type(c, t)$. A connection is represented by literal $conn\ (c1, p1, c2, p2)$ where $p1$ (resp. $p2$) is a port of component $c1$ (resp. $c2$). Usually an attribute value v assigned to attribute a of component c is represented by a literal $val(c, a, v)$. In our example, we omit val - predicates to keep the presentation short. The configuration knowledge of each of the three involved configurators is defined by a domain description (DD) comprising sets of logical sentences that specify compatibility constraints and the derivation of additional facts. In addition to the constraints C_i listed below, a set of application independent sentences denoted by C_{basic} is included in the domain description, specifying that connections are symmetric, that a port can only be connected to one other port, and that components have a unique type.

$DD_{switch} = \{C_1,\ C_2\} \cup C_{basic}$.
$DD_{ip} = \{C_3,\ C_4\} \cup C_{basic}$. $DD_{msg} = \{C_5\} \cup C_{basic}$.

C_1 : *"If the switch has more than 200 end devices then a large rack is needed."*
$\forall T, C : type(T, tecom) \wedge devices(T, C) \wedge$
$C > 200 \Rightarrow \exists L :$
$type(L, lrack) \wedge conn(L, tecom, T, rack)$.

C_2 : *"If the customer requires voice-over-ip then the ipvoice application must be installed."*
$\forall T : type(T, tecom) \wedge voice\text{-}over\text{-}ip(T) \Rightarrow \exists I :$
$type(I, ipvoice) \wedge conn(I, tecom, T, ipvoice)$.

C_3 : *"An ipvoice application consists either of a swpack1 or swpack2 software module."*
$\forall I : type(I, ipvoice) \Rightarrow \exists P :$
$(type(P, swpack1) \vee type(P, swpack2)) \wedge$
$conn(P, ipvoice, I, swpack)$.

C_4 : *"A swpack1 software module is incompatible with upgrade uppack."*
$\forall T, I, M, P1, U : type(T, tecom) \wedge type(I, ipvoice) \wedge$
$conn(I, tecom, T, ipvoice) \wedge type(M, msger) \wedge$
$conn(M, tecom, T, msger) \wedge type(P1, swpack1) \wedge$
$conn(P1, ipvoice, I, swpack) \wedge type(U, uppack) \wedge$
$conn(U, msger, M, upgr) \Rightarrow false$.

C_5 : *"If the software msger is sold together with the ipvoice application then it must contain the upgrade uppack."*
$\forall T, M, I : type(T, tecom) \wedge type(M, msger) \wedge$
$conn(M, tecom, T, msger) \wedge type(I, ipvoice) \wedge$
$conn(I, tecom, T, ipvoice) \Rightarrow \exists P :$
$type(P, uppack) \wedge conn(P, msger, M, upgr)$.

In our domain a system requirements specification (SRS) provided by the customer is only sent to the switching hardware manufacturer. It is a logic theory that comprises predicates from $CONL$ as well as any other predicates that specify the requirements a customer wants to be fulfilled.

$$SRS_{switch} = \{\exists T, M : type(T, tecom) \wedge \\ devices(T, 300) \wedge voice\text{-}over\text{-}ip(T) \wedge \\ type(M, msger) \wedge conn(M, tecom, T, msger).\}$$

Given the above constraints and customer requirements, central problem solving would achieve the following complete and consistent configuration result[1]:

$$CONF = \{type(id_1, tecom).\ type(id_2, lrack). \\ type(id_3, ipvoice).\ type(id_4, msger). \\ type(id_5, swpack2).\ type(id_6, uppack).\}$$

Note however, that a central approach is not feasible for security and privacy concerns of involved business entities and the question is how to solve this task for the distributed case. Therefor, we aim at defining the distributed configuration problem and at stating the conditions under which the distributed solving generates equivalent solutions to a central approach.

3 Formalizing Distributed Configuration

In the general framework of [2], a configurator knowledge base consists of a set of logical sentences DD describing available component types, their attributes and connection points as well as constraints on legal product constellations. As sketched in Section 2, configuration problems are solved according to a system requirements specification SRS and the configuration result can be described by means of a set of positive ground literals using predicate symbols from $CONL$.

3.1 Central Configuration Approach

Definition (Configuration problem): *A configuration problem is described by a triple $(DD, SRS, CONL)$, where DD and SRS are sets of logical sentences and $CONL$ is a set of predicate symbols. DD represents the domain description, SRS the system requirements specification for a configuration problem instance. A configuration $CONF$ is described by a set of positive ground literals[2] whose predicate symbols are in $CONL$.*$\square$

[1] For reasons of presentation, we employ only $type/2$ predicates for representing configurations and omit the $conn/4$ predicates

[2] By using Skolem constants we have decoupled the representation of a configuration solution from the problem description. Therefor validity of configurations is independent of a bijective renaming of these constants.

Definition (Consistent configuration): *Given a configuration problem $(DD, SRS, CONL)$, a configuration $CONF$ is consistent iff $DD \cup SRS \cup CONF$ is satisfiable.* $\square$

To ensure the completeness of a configuration, additional formulae for each symbol in $CONL$ have to be introduced to $CONF$, e.g., for the *type* predicate:

$$type(X, Y) \Rightarrow type(X, Y) \in CONF.$$

We denote the configuration $CONF$ extended by these axioms with $\widehat{CONF}$.

Definition (Valid and irreducible configuration): *Let $(DD, SRS, CONL)$ be a configuration problem. A configuration $CONF$ is valid iff $DD \cup SRS \cup \widehat{CONF}$ is satisfiable. $CONF$ is irreducible if there exists no other valid configuration $CONF^{sub}$ such that $CONF^{sub} \subset CONF$.* $\square$

3.2 Distributed Configuration Approach

Definition (Distributed configuration problem): *A distributed configuration problem for n different configuration agents is described by a triple $(DD_{set}, SRS_{set}, CONL)$ where*

$DD_{set} = \{DD_1, \ldots, DD_n\}$ *and*

$SRS_{set} = \{SRS_1, \ldots, SRS_n\}$.

Each element of DD_{set} and of SRS_{set} is a set of logical sentences and $CONL$ is a set of predicate symbols. For $k \in \{1, \ldots, n\}$, DD_k corresponds to the domain description of the configuration system k and SRS_k specifies its system requirements. A configuration $CONF$ is described by a set of positive ground literals whose predicate symbols are in $CONL$.$\square$

Remark: In extension to the introductory example, all configuration agents can be initialized with an individual system requirements specification contained in the set SRS_{set}.

Definition (Valid solution to a distributed configuration problem): *Given a distributed configuration problem $(DD_{set}, SRS_{set}, CONL)$, a configuration $CONF$ is valid iff $DD_k \cup SRS_k \cup \widehat{CONF}$ is satisfiable $\forall k \in \{1, \ldots, n\}$.*$\square$

In practice configurators of suppliers collaborate by exchanging (partial) configurations, i.e., these configurators can be seen as independent modules jointly constructing a common solution. Related to our case this implies that the domain descriptions $DD_1 \ldots DD_n$ and the system requirements $SRS_1 \ldots SRS_n$ of each configurator are independent except assertions regarding the configuration. We achieve this property by using disjoint sets of predicate symbols in each DD_k and SRS_k but allow the joint use of predicate symbols contained in $CONL$, i.e., for every pair of configurators i, j

$(psymbols^3(DD_i) \cup psymbols(SRS_i)) \cap (psymbols(DD_j) \cup psymbols(SRS_j)) = X$, where $X \subseteq CONL$. This way, dependencies, that surpass the knowledge of local companies, are considered via their effects on the configuration result. We call this property *defined interfacing*.

Theorem: *Let $(DD, SRS, CONL)$ be a configuration problem and $(DD_{set}, SRS_{set}, CONL)$ a distributed configuration problem with defined interfacing where*
$DD = \bigcup_{dd \in DD_{set}} dd$ and
$SRS = \bigcup_{srs \in SRS_{set}} srs$.
$CONF$ is a valid configuration for $(DD, SRS, CONL)$ iff $CONF$ is a valid solution for the distributed configuration problem $(DD_{set}, SRS_{set}, CONL)$.$\Box$

Proof (sketch):
$(\Rightarrow)$ Since $DD \cup SRS \cup \widehat{CONF}$ is satisfiable, and $DD_k \subseteq DD$, $SRS_k \subseteq SRS$ also $DD_k \cup SRS_k \cup \widehat{CONF}$ is satisfiable. It follows $CONF$ is also a valid solution for the distributed configuration problem $(DD_{set}, SRS_{set}, CONL)$. $\Box$

$(\Leftarrow)$ $DD_k \cup SRS_k \cup \widehat{CONF}$ (we call this theory T_k) and $DD_j \cup SRS_j \cup \widehat{CONF}$ (called T_j) with $k \neq j$ are consistent. Let us assume $DD_k \cup SRS_k \cup \widehat{CONF} \cup DD_j \cup SRS_j$ is inconsistent. It follows that $T_k \vdash \neg(DD_j \cup SRS_j)$. The theory $(DD_j \cup SRS_j)$ can be transformed to an equivalent theory expressed by a set of clauses Cs_j. Consequently, T_k has to imply the negation of a clause C_{CONL} where C_{CONL} follows from Cs_j. Note, that T_k can only imply such a clause $\neg C_{CONL}$ which solely consists of predicates of CONL since T_k and T_j have only predicates in common which are in CONL. Because $\widehat{CONF} \subseteq T_k$ is a complete theory w.r.t. predicates in CONL it follows that $\widehat{CONF} \vdash \neg C_{CONL}$. However, $DD_j \cup SRS_j$ implies C_{CONL} and therefore T_j is inconsistent which is a contradiction to the fact that T_j is consistent. Consequently, $DD_k \cup SRS_k \cup \widehat{CONF} \cup DD_j \cup SRS_j$ is consistent. By applying this argument to all elements of DD_{set} and SRS_{set} it follows that $\bigcup_{dd \in DD_{set}} dd \bigcup_{srs \in SRS_{set}} srs \cup \widehat{CONF}$ is consistent. $\Box$

Corollary: *Let $(DD, SRS, CONL)$ be a configuration problem and $(DD_{set}, SRS_{set}, CONL)$ a distributed configuration problem with defined interfacing where*
$DD = \bigcup_{dd \in DD_{set}} dd$ and
$SRS = \bigcup_{srs \in SRS_{set}} srs$.
A valid configuration $CONF$ for $(DD, SRS, CONL)$ is irreducible iff $CONF$ is a valid solution to the distributed configuration problem $(DD_{set}, SRS_{set}, CONL)$ and there exists no other valid solution $CONF^{sub}$ to the distributed configuration problem such that $CONF^{sub} \subset CONF$. $\Box$

[3] The function $psymbols(T)$ returns all predicate symbols that are employed in the logical theory T.

3.3 Conflicts

When solving a configuration problem, partial solutions are extended with the goal to generate valid configurations. During the problem solving phase it could be discovered that such partial solutions are in conflict with $DD \cup SRS$. As a consequence these conflicts must guide the subsequent search process in order to avoid the rediscovery of inconsistent configurations. Due to the *defined interfacing* property, conflicts can only be caused by predicate symbols from $CONL$ in $CONF$, i.e.:

Definition (Conflict): *Let $(DD, SRS, CONL)$ be a configuration problem and $CONF$ be a consistent set of sentences in $CONL$. $CONF$ is a conflict of $(DD, SRS, CONL)$ iff $SRS \cup DD \vdash \neg CONF$.* $\square$

The relation between conflicts and configurations is described as follows.

Theorem: *Let $(DD, SRS, CONL)$ be a configuration problem, $NG-CONFLICTS$ is its set of negated conflicts and $\widehat{CONF}$ be a configuration including the completeness axioms. $\widehat{CONF}$ is a valid configuration iff $\widehat{CONF} \cup NG - CONFLICTS$ is satisfiable.* $\square$

Proof (sketch):

$(\Rightarrow)$ Since $DD \cup SRS \cup \widehat{CONF}$ is satisfiable and $NG - CONFLICTS$ is entailed by $DD \cup SRS$ it follows that $\widehat{CONF} \cup NG - CONFLICTS$ is satisfiable. $\square$

$(\Leftarrow)$ Let $\widehat{CONF}$ be a configuration where $\widehat{CONF} \cup CONFLICTS$ is satisfiable. Suppose $\widehat{CONF}$ is not a valid configuration i.e. $DD \cup SRS \cup \widehat{CONF}$ is unsatisfiable. Therefore, $DD \cup SRS \vdash \neg\widehat{CONF}$. But then $\widehat{CONF}$ would be a conflict and $\neg\widehat{CONF}$ must be included in $NG - CONFLICTS$, contradicting the fact that $\widehat{CONF} \cup NG - CONFLICTS$ is satisfiable. $\square$

Definition (Minimal conflict): *Let $(DD, SRS, CONL)$ be a configuration problem and CONF a conflict. CONF is a minimal conflict of $(DD, SRS, CONL)$ iff for all conflicts $CONF^{sub}$ either $\neg CONF^{sub} \not\models \neg CONF$ or $\neg CONF^{sub} \equiv \neg CONF$.* $\square$

Note, that when searching for valid configurations we must achieve consistency with the negated conflicts. Since the negation of minimal conflicts implies the negation of non-minimal conflicts, thus a non-minimal conflict needs not to be considered. The same argument holds for an equivalent conflict. In the following we relate conflicts of the central and the distributed configuration approach.

Corollary: *Let $(DD, SRS, CONL)$ be a configuration problem, $\widehat{CONF}$ be a configuration including the completeness axioms, and $(DD_{set}, SRS_{set}, CONL)$ a distributed configuration problem with defined interfacing where*

$DD = \bigcup_{dd \in DD_{set}} dd$ *and*

$SRS = \bigcup_{srs \in SRS_{set}} srs.$

$\widehat{CONF}$ is a conflict for $(DD, SRS, CONL)$ iff there exists a $k \in \{1, \ldots, n\}$ s.t. $\widehat{CONF}$ is a conflict for $(DD_k, SRS_k, CONL)$. $\square$

Note, that every conflict found by a local configuration agent is a conflict for the complete (central) configuration problem. These conflicts must be communicated among the agents, in order to ensure that superfluous work on conflicting configurations is avoided.

4 Basic Model for Interaction

In order to show the feasibility of distributed configuration problem solving according to the above definitions we outline an architectural setting of cooperating agents and propose an algorithm for interaction. Note however, that this model represents a theoretical framework for the general case. For our concrete implementation in an application-oriented international research project we exploit domain specifics of configuration problem solving as described in the next subsection *Extensions for Efficiency*. The configuration knowledge is distributed over a set of *n* configuration agents which may configure concurrently. The communication among them is coordinated via a facilitator agent that collects the (partial) configurations from each agent and distributes them among the others. So there is no direct communication between configurators, but only indirect via the facilitator. This architecture is chosen because it poses less requirements on the capabilities of configuration agents than *peer-to-peer* communication, where each agent must be able to distinguish between several communication channels. As soon as a configuration agent detects a conflict with the joint configuration, the others are informed and measures for conflict resolution are taken. This resolution strategy ensures that a conflict never occurs twice during a session and that the non-existence of a valid configuration for the overall task is detected. We do propose a very general negotiation strategy for conflict resolution, because we consider the integration of already existing configuration systems into this framework.
The configurator agents communicate only with the faciliator, where the exchanged messages have the following signatures:

- $request_k^{no}(CONF^{s_i})$: The configurator *k* receives the configuration $CONF^{s_i}$ and checks if is locally satisfiable. s_i denotes the search depth of the algorithm for this intermediate configuration solution and no counts the interaction cycles.
- $reply_k^{no}(CONF_k^{s_i+1})$: The configurator *k* communicates the configuration result $CONF_k^{s_i+1}$ in reply to $request_k^{no}(CONF^{s_i})$ back to the facilitator. $CONF_k^{s_i+1}$ is a valid local configuration of configurator *k*.
- $conflict_k(CONF^{s_j})$: With this message the configurator *k* alerts the facilitator, that $CONF^{s_j}$ is not satisfiable with its local knowledge base.
- $add\text{-}conflict(C)$: Once the facilitator was alerted with a $conflict$ message, it broadcasts this conflict C to all configuration agents that is then negated and added to their local system requirements SRS_k.

The **facilitator agent** initially distributes only the non-empty sets of individual system requirements SRS_k to the configuration agents that store them - see Algorithm (a). Obviously, requesting a configuration from a configurator without any system requirements would produce merely incidental results and is therefor avoided. Then the facilitator starts the problem solving process by broadcasting $request_k^1(\{\})$ to each recipient of a non-empty SRS_k and awaits $reply_k^1(CONF_k^{s_1})$ messages from these

configuration agents, only (b.2). After collection of replies (b.2) the facilitator unifies the locally completed configurations and broadcasts them to all configuration agents with a $request_k^{no}(\bigcup_k CONF_k^{s_i})$ message. This is now possible because the intermediate result $CONF^{s_i}$ restricts the further solution search of agents. In case at least one of the remote configurators replies a $conflict_k(CONF^{s_j})$ message the facilitator initiates the conflict resolution strategy (c). Here we chose a strategy where it is in the responsibility of every single agent of not delivering a conflicting configuration twice. This is achieved because the facilitator communicates the conflicting configuration $CONF^{s_j}$ to all agents via an $add\text{-}conflict(CONF^{s_j})$ message and backtracks by demanding another reply to $request_k^{no}(CONF^{s_{j-1}})$ (c.1). All replies to the previous request are discarded (b.1). The algorithm terminates either with a valid solution or detects that there exists no solution (c.2). The latter case is implied if the empty set is not satisfiable with the local knowledge base including stored conflicts. A valid configuration for the overall configuration task is found, when no configurator has added new ground facts to $CONF^{s_i}$ during an interaction cycle (b.4), i.e., $CONF^{s_i} = CONF^{s_{i+1}}$.
When a **configuration agent** receives a $request_k^{no}(CONF^{s_i})$ message the algorithm distinguishes between two problem solving levels (d).

First the satisfiability of the received configuration is tested.

Definition (Local satisfiability): *Local satisfiability for agent k is given, iff configuration $CONF^{s_i}$ is satisfiable with its local knowledge base: $DD_k \cup SRS_k \cup CONF^{s_i}$ is satisfiable.*

If *Local Satisfiability* is given, the configuration agent completes the initial configuration $CONF^{s_i}$ to a locally valid configuration $CONF_k^{s_{i+1}}$ (d.1), which is performed in the algorithm by the function $configure$.

Definition (Local validity): *Local validity for agent k is given, iff configuration $CONF_k^{s_{i+1}}$ is a valid configuration w.r.t. its local knowledge base: $CONF_k^{s_{i+1}}$ is valid for $(DD_k, SRS_k, CONL)$.*

If *Local Satisfiability* is not given, the configuration agent replies with a $conflict_k$ $(CONF^{s_i})$ message to the initial request $request_k^{no}(CONF^{s_i})$ (d.2).

When a configurator receives a $add\text{-}conflict(C)$ message it stores it by expanding its local system requirements (e):

$$SRS_k^{'} = SRS_k \cup \neg C.$$

At this stage redundant clauses (e.g., non-minimal conflicts) can be removed from SRS_k.

```
**Algorithm -** *Behaviour of facilitator agent*
(a) initialize($SRS_{set}$) do
      $\forall SRS_k \neq \{\} : \ forward\ SRS_k$ to agent k;
      $CONF^{s_0} = \{\}$; $count = 1$;
      $\forall SRS_k \neq \{\} : \ send(request_k^1(CONF^{s_0}))$;
    end do;
(b) when received($reply_k^{no}(CONF_k^{s_i})$) do
(b.1) if $no = count$ then
        $CONF^{s_i} = CONF_k^{s_i} \cup CONF^{s_i}$;
  (b.2) if $(\forall k\{\in 1,\ldots,n\} : \ received(CONF_k^{s_i}))\vee$
        $((count = 1)\wedge$
        $(\forall SRS_k \neq \{\} : \ received(CONF_k^{s_i})))$ then
    (b.3) if $CONF^{s_i} \neq CONF^{s_{i-1}}$ then
            count++;
            $\forall k \in \{1,\ldots,n\} : \ send(request_k^{count}(CONF^{s_i}))$;
    (b.4) else
            terminate algorithm, out: $CONF = CONF^{s_i}$;
          end if
        end if
      end if
    end do;
(c) when received($conflict_k(CONF^{s_j})$) do
(c.1) if $CONF^{s_j} \neq \{\}$ then
        $\forall k \in \{1,\ldots,n\} : \ send(add\text{-}conflict(CONF^{s_j}))$;
        count++;
        $\forall k \in \{1,\ldots,n\} : \ send(request_k^{count}(CONF^{s_{j-1}}))$;
(c.2) else
        terminate algorithm, out: no configuration exists;
      end if
    end do;

Algorithm - Behaviour of configuration agent
(d) when received($request_k^{no}(CONF^{s_i})$) do
(d.1) if $locally\ satisfiable(DD_k \cup SRS_k \cup CONF^{s_i})$ then
        $CONF_k^{s_{i+1}} = configure(CONF^{s_i})$;
        $send(reply_k^{no}(CONF_k^{s_{i+1}}))$;
(d.2) else
        $send(conflict_k(CONF^{s_i}))$;
      end if
    end do;
(e) when received( $add\text{-}conflict_k(C)$ ) do
      $SRS_k = SRS_k \cup \neg C$;
    end do;
```

4.1 Extensions for Efficiency

Configuration problems have the property to be usually underconstrained and there exist many good solutions that can be accepted from the standpoint of a domain expert. Furthermore, similar to a centralized approach, heuristics exist to guide the solution search within and between the configuration agents. These allow us to avoid an inefficient *blind* search of the solution space and provide optimized solutions according to some criteria such as price or quality. For design of our prototype implementation we identified the following approaches:

- In a realistic economic setting one destined configuration agent will act as a main vendor that also fulfills the task of the facilitator. Further some partial sequentialization between configurators can be assumed, i.e., the main manufacturer or service provider will configure locally as far as possible and restrict this way the solution space of its suppliers. When reducing the degree of parallelism of solution search the probability for conflict occurence can be obviously diminished. Restricting concurrent solution search to agents whose configuration results do not have side-effects on each other is an additional heuristic. Further a partial ordering of configurators can be used for an advanced negotiation strategy for conflict resolution, where agents with lower priority are the first ones to repair their configuration results. Such a scenario where all configuration agents sequentially add new predicates to the calculated configuration of the predecessor in a supply chain is a specific instantiation of the more general model presented in this paper.
- When configuring complex telecommunication systems computed configurations tend to become quite large with $CONF$ encompassing thousands of facts. It is obvious that not all components and connections of the switching node are of interest to the configuration agent that determines the configuration of the add-on product. Therefor measures towards intelligent filtering of the message content need to be taken. We can assume that configuration knowledge is not randomly partitioned. Based on this partitioning of configuration knowledge, we are able to identify the vocabulary of product domain concepts that specifies the configuration capabilities and informational interest on components and connections for which it has constraints defined in its local knowledge base. By employing domain ontologies we can give an abstract description of the product domain of the configuration agent. Therefor only those facts of the overall configuration solution need to be communicated to a specific configuration agent that correspond to its domain ontology.
- A further step towards lower space complexity is the reduction of conflict size. This can be achieved, if configuration agents are capable of generating minimal conflicts following the definition in Section 3.3. Techniques from model based diagnosis can be exploited to improve conflict generation.

4.2 Solving the Example

In the following we show how to solve the example from Section 2 with our algorithm. In the example three agents are involved, i.e., $k \in \{switch, ip, msg\}$. The problem solving phase starts when the facilitator agent forwards the system requirements SRS_{switch}

to the switching hardware manufacturer and initiates the solution search by sending $request^1_{switch}(\{\})$. Therefor the agent *switch* is the only one to reply to the facilitator in the first cycle of interaction:

(1) $reply^1_{switch}(\{type(id_{s1}, tecom).\ type(id_{s2}, lrack).$
$type(id_{s3}, ipvoice).\ type(id_{s4}, msger).\})$

The facilitator distributes the received configuration as $CONF^{s_1}$ to all agents.

(2) $reply^2_{switch}(CONF^{s_1} \cup \{\})$
$reply^2_{ip}(CONF^{s_1} \cup \{type(id_{i1}, swpack1).\})$
$reply^2_{msg}(CONF^{s_1} \cup \{type(id_{m1}, uppack).\})$

The facilitator unifies the received partial configurations and broadcasts a $request^3_k$ $(CONF^{s_2})$ to all agents.

(3) $conflict_{ip}(CONF^{s_2})$ *because of Constraint* C_4.

The facilitator discards the $reply^2_k$ messages from agents *switch* and *msg*. It broadcasts $add\text{-}conflict(CONF^{s_2})$ and afterwards backtracks with $request^4_k(CONF^{s_1})$ to all agents.

(4) $reply^4_{switch}(CONF^{s_1} \cup \{\})$
$reply^4_{ip}(CONF^{s_1} \cup \{type(id_{i2}, swpack2).\})$
$reply^4_{msg}(CONF^{s_1} \cup \{type(id_{m2}, uppack).\})$

The facilitator generates the union set of all received partial configurations and broadcasts them for another cycle of interaction. Now, no one detects a conflict. All configuration agents determine the validity of the configuration and do not need to derive additional facts. Therefor the algorithm terminates with the same solution as with central problem solving.

4.3 Analysis

For analysis we employ the basic assumption that all configuration agents are capable of generating valid configuration results and are complete w.r.t. the set of all valid configurations, which we assume to be limited for practical reasons. This can be achieved by limiting the number of possible components in an artifact. In order to show the *soundness* of the algorithm we must show that each generated solution $CONF$ satisfies the criteria of a valid distributed configuration stated in section 3. This is given, because $\forall k \in \{1, \ldots, n\} : DD_k \cup SRS_k \cup \widehat{CONF}^{s_i}$ is satisfiable and $CONF^{s_i} = CONF$. For proofing the *completeness* of the algorithm we must show that if a solution exists the algorithm terminates with a configuration solution $CONF$, otherwise the algorithm terminates with a failure indication. Let us first assume that the algorithm terminates. This happens either by giving a correct solution, or terminating, because no solution exists: $\nexists CONF : DD_k \cup SRS_k \cup \widehat{CONF}$ *is satisfiable* $\forall k \in \{1, \ldots, n\}$. Finally we have to show the algorithm terminates. Sources for infinite processing loops are cycles in message passing and subsequent generation of the same conflict. Infinite processing loops are not possible because all agents can only receive requests from the facilitator that are replied either by a $reply^{no}_k$ or a $conflict_k$ message, and the number of these messages is restricted due to the initial assumption of a limited solution space. Subsequent generation of the same conflict is avoided, because inconsistent configurations are distributed as conflicts[4] to all agents via $add\text{-}conflict$ messages and locally stored by them. As all valid configurations are consistent with the set of already negated conflicts, no conflicting configuration is produced twice by a configuration agent. Further with this algorithm all valid configurations can be found. If an already found solution is negated and added to the system requirements, a new search can be initiated and the algorithm will find another solution that is different from the previous one. As there exists only a finite number of configurations, subsequently all solutions will be found.

5 Aspects of Integration

In order to be integrated in the presented architectural setting, a configurator must satisfy the following requirements resulting from the protocol:

– Support of $request_k^{no}$ requests: The configurator must accept a partial configuration as starting point for configuration problem solving and generate a complete configuration solution.
– Support for $add\text{-}conflict$ messages: The configurator must at least be able to compute alternative solutions to fulfill this requirement. An agent wrapper would then store the received conflicts and request alternative solutions from the native configurator interface, until all stored negated conflicts are satisfied.

The latter requirement can not be met by pure rule-based configurators, which always calculate only exactly one solution for given requirements. Furthermore, different expressiveness of proprietary knowledge representations may pose a problem. Employing bridging rules, that map between different representation concepts are always imperfect in the sense that they are heuristic.
A different issue is the process of distributedly creating and maintaining configuration knowledge. There must be some guidance provided and the consistency of the knowledge bases has to be assured. Having different representation mechanisms, a shared ontology must be adopted, that provides a common view on the product domain.
Currently, separate sets of initial requirements SRS_k are assumed. However, the choice among similar products of different companies is another point to be addressed and some form of reasoning on the selection of a supplier has to be introduced.

6 Related Work

There is a long history in developing configuration tools in knowledge-based systems. Progressing from rule-based systems higher level representation formalisms were developed, such as various forms of constraint satisfaction [4], or description logics [8]. However there is no support for integrating these systems in order to allow cooperative configuration.
When using a constraint-based approach for configuration tasks, several techniques

[4] Note that Skolem constants in calculated configurations are converted to all-quantified variables, if the conflict is negated.

for distributed problem solving have been proposed. For problem representation a distributed CSP is proposed in [13] and several algorithms for problem solving such as an *asynchronous backtracking*, an *asynchronous weak-commitment* search or a *distributed breakout* algorithm are presented. However, configuration tasks are more dynamic in nature and therefor a CSP representation, where all problem variables must be known from the beginning, is not appropriate in many application domains. Dynamic constraint satisfaction is more suitable for representing and solving such synthesis tasks [9] [12], because the set of problem variables may vary according to some *activity contraints*. In [3] a distributed dynamic CSP is defined and a modification of the asynchronous backtracking algorithm from [13] is applied for problem solving. When configuring large technical systems, the limitation of a dynamic CSP representation (the amount of maximally active variables must be known from the beginning) becomes evident and a generic CSP representation [4] [7] has been proposed. There, new instances of problem variables can be created from meta-variables during problem solving. However, no representation that allows the distribution of knowledge over several agents has been presented so far.

The proposed architecture for distributed configuration relates to previous research projects such as TSIMMIS [5] or Infomaster [6]. They provide an integrated access to multiple distributed heterogenous information sources on the Internet. Our approach for distributed configuration goes a step further, because not only information sources but problem-solving agents with local knowledge are integrated, thus giving the illusion of a centralized, homogenous configuration system.

In the area of distributed configuration-design problem solving [1] proposed an agent architecture. The aim of this work is to find a concurrent problem solving process in order to improve efficiency, whereas our concern is to provide effective support of distributed configuration problem solving, where knowledge is already distributed between different agents.

7 Conclusions

Due to internet technologies, business processes cross enterprise boundaries, which boosts the demand for distributed problem solving methods. In the domain of product configuration the integration of Web-based configuration agents is necessary in order to match the needs that arise from temporary cooperation between highly specialized business entities. In this paper we defined a general consistency-based approach towards the joint provision of configuration solutions by multiple configurators. Based on a formal definition of the distributed configuration approach, it was shown under which conditions distributed configuration problem solving produces equivalent results to the central case. Partial configurations which are in conflict to the system requirements and domain descriptions facilitate the search process. The concept of conflicts was introduced and its relation to valid configurations shown. Further a complete and sound algorithm for cooperation was presented which allows the integration of domain dependent heuristics.

Acknowledgments. This work was partly funded by the EC under contract no. IST-1999-10688 and the Austrian Federal Ministry of Education, Science and Culture.

References

1. T.P. Darr and W.P. Birmingham. An Attribute-Space Representation and Algorithm for Concurrent Engineering. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 10(1):21–35, 1996.
2. A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based Diagnosis of Configuration Knowledge Bases. In *Proceedings of the 14th ECAI*, pages 146–150, Berlin, Germany, 2000.
3. A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. Distributed Configuration as Distributed Dynamic Constraint Satisfaction. In *Proceedings of the 14th IEA/AIE*, pages 434–444, Budapest, Hungary, 2001.
4. G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner. Configuring Large Systems Using Generative Constraint Satisfaction. In B. Faltings and E. Freuder, editors, *IEEE Intelligent Systems, Special Issue on Configuration*, volume 13,4, pages 59–68. 1998.
5. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.
6. M. Genesereth, A. Keller and O. Duschka. Infomaster: An Information Integration System. In *Proceedings of the 23rd ACM SIGMOD International Conference on Management of Data*, pages 539–542, Tucson, Az, USA, 1997.
7. D. Mailharro. A classification and constraint-based framework for configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Special Issue: Configuration design*, 12(4):383–397, 1998.
8. D.L. McGuiness and J.R. Wright. Conceptual Modeling for Configuration: A Description Logic-based Approach. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Special Issue: Configuration design*, 12(4):333–344, 1998.
9. S. Mittal and B. Falkenhainer. Dynamic Constraint Satisfaction Problems. In *Proceedings of AAAI 1990*, pages 25–32, Boston, MA, 1990.
10. S. Mittal and F. Frayman. Towards a Generic Model of Configuration Tasks. In *Proc. of the 11th IJCAI*, pages 1395–1401, Detroit, MI, 1989.
11. D. Sabin and R. Weigel. Product Configuration Frameworks - A Survey. In E. Freuder B. Faltings, editor, *IEEE Intelligent Systems, Special Issue on Configuration*, volume 13(4), pages 50–58. 1998.
12. T. Soininen, E. Gelle and I. Niemelä. A Fixpoint Definition of Dynamic Constraint Satisfaction. In *5th International Conference on Principles and Practice of Constraint Programming - CP'99*, pages 419–433, Alexandria, USA, 1999.
13. M. Yokoo. *Distributed Constraint Satisfaction - Foundations of Cooperation in Multi-agent Systems*. Springer, Berlin, Germany, 2001.