Applying Constraint Programming to Incorporate Engineering Methodologies into the Design Process of Complex Systems

Odellia Boni and **Fabiana Fournier**
and **Nir Mashkif** and **Yehuda Naveh**
and **Aviad Sela** and **Uri Shani**
IBM Research - Haifa, Israel
odelliab,fabiana,nirm,naveh,sela,shani@il.ibm.com

Zvi Lando and **Alon Modai**
Israel Aerospace Industries Ltd.
zlando,amodai@iai.co.il

Abstract

When designing a complex system, adhering to a design methodology is essential to ensure design quality and to shorten the design phase. Until recently, enforcing this could be done only partially or manually. This paper demonstrates how constraint programming technology can enable automation of the design methodology support when the design artifacts reside in a central repository. At any phase of the design, the proposed constraint programming application can indicate whether the design process data complies with the methodology and point out any violations that may exist. Moreover, the application can provide recommendations regarding the design process. The application was successfully used to check the methodology conformance of an industrial example and produced the desired outputs within reasonable times.

1 Introduction

The design of complex systems is a complicated process involving many engineers of various disciplines who use numerous tools from different vendors. These tools are used to describe the system's decomposition into subsystems and to manage the design artifacts belonging to each of those subsystems. Usually, each tool can manage only the design artifacts belonging to a single discipline. However, there exist relations between design artifacts belonging to different disciplines and hence, residing in different tools.

Systems engineering methodologies include guidelines that constrain the relations between the design artifacts, taking into account the subsystems decomposition, in order to maintain traceability in the design data. However, current methods for checking the conformance to those guidelines are limited or error-prone. The reasons for that are two-fold. The first is the difficulty to manage and present the relations net. Some parts of this net are managed in different tools while other parts are managed manually. The second reason is that the current means used for checking could check only local or simple guidelines, one at a time.

Lately, a new framework which can manage the relations net between the design artifacts of the modeling phase was proposed (Gery, Modai, and Mashkif 2010). In this paper we present a novel way to check the conformance of the relations net managed by this framework to the methodology guidelines. We check all the guidelines, complex and global guidelines as well as local and simple, simultaneously. To this end, we develop a concise graphic representation of the system's design data and formulate the methodology guidelines as rules on this graph structure. This allows us to describe the problem of enforcing systems engineering methodology as a rules-checking problem. We built an application based on constraint programming, which can check whether a graph describing partial data of the system contains violations of the rules or contradictions (i.e., cannot be extended into a graph that satisfies the rules). The application also lists the violations or analyzes the source of contradiction. If no violations or contradictions are found, the application provides information on how the graph can be extended without violating the rules.

The paper is organized as follows: In Section 2 we provide some background on systems engineering methodology, and explain the desired support using a simple example. In Section 3 we describe how the problem of checking the methodology guidelines can be modeled as a constraint programming (CP) problem and how a CP solver can be utilized to achieve the desired methodology support. In Section 4 we show results of running a CP-based application on a tutorial example. In Section 5 we discuss the advantages and disadvantages of checking the systems engineering methodology using CP. In Section 6 we summarize and draw conclusions.

2 Background

2.1 Systems Engineering

Complex systems are usually composed of embedded software, physical and electric components interacting with each other. Such systems are common to the automotive, aerospace and defense, power, and water industries, in which any errors in the design phases of the system can cause large costs and delays. To cope with these challenges, projects in these industries are highly model-oriented. Engineers use tools to model the different aspects of the product, from inception to parts manufacturing.

Within this wide span of activities, our work concentrates on the high level model-based design and testing phases, which involve many disciplines including requirements engineering, functional analysis, and verification.

Each of these disciplines uses a different modeling scheme to manage the data and incorporates some commonality and similarities in the design artifacts it embraces. Hence, the design artifacts can be categorized into types (e.g., requirement, function, test) where artifacts of the same type belong to the same discipline and share similar attributes. Since the modeling scheme varies from discipline to discipline, different tools are often used to design and manage the artifacts belonging to each discipline.

There are, however, relations between the design artifacts from different disciplines. These relations express the design logic such as 'this requirement is implemented by this system function'.

Another feature of complex systems is structural hierarchy, which decomposes the system into its building blocks, with inputs and outputs streaming between them. Each of these sub-components is regarded as a complex system of its own having its own design artifacts and is decomposed in turn.

Hence, the system's description is complex from two aspects. There are many disciplines interacting within a single system level and each discipline is coherently decomposing all its information into sub-systems.

2.2 Systems Engineering Methodology

To cope with the challenges in complex systems design, systems engineering methodologies have been developed (Engel 2010). Many of these methodologies provide guidelines regarding the design process and logic to improve traceability and to facilitate information exchange between tools of different disciplines. The methodologies aim to ensure design consistency and conformance to the system's requirements. We suggest that the methodologies be represented as rules to which the structure of the relations between the artifacts should adhere.

2.3 Current Methodology Support

As mentioned earlier, the design work in each discipline is often carried out by a different tool. Many of the tools cannot cope with the relations to artifacts residing in the other tools. Hence, today, the process of checking inter-tool relations is either completely manual or only partially automated.

For the manual checking, the tools produce documents. These documents are processed manually by engineers to ensure consistency among the documents. This method involves high costs in manual labor and has a greater probability of introducing errors. Often document management tools are used to monitor this process.

Automated checking utilizes new methods that were developed to exchange data between tools - especially those with high inter-dependency (e.g., design and simulation tools) (Holt and Perry 2008). These methods use a shared model scheme to enable importing data into the tool in which the checking takes place. In most industries only peer-to-peer tool integration is performed, hence only relations between artifacts residing in at most two tools at a time can be checked. With today's systems employing numerous tools, even within the same company, this peer-to-peer situation covers only a small portion of the relations net.

Automated checking of the relations structure within a tool done today is limited also with regard to the rules. It usually takes place using queries or scripts, which need to be customized for each rule. Moreover, each rule is checked separately without considering any mutual impact between the rules. There are tools which can perform automatic checking of local methodology rules whenever any data is refreshed, modified, or added. Continuous checking of such local rules is performed, for instance, by the Eclipse Modeling Framework (Steinberg et al. 2008) toolkit on the Eclipse platform, on eCore models. Yet, the most valuable rules are those dealing with the global consistency of the integrated model, such as the cycle rules described in Section 3.3, which are beyond the scope of such tools.

2.4 Proposed Methodology Support

Recently, a new approach to tools interoperability evolved, known as Open Services for Lifecycle Collaboration (OSLC) [1]. OSLC takes the open Internet web experience into the modeling tools realm. Any artifact managed by a tool is considered a resource, similar to any other resource on the Internet. By having its own unique ID, referred to as a Uniform Resource Identifier (URI), the resource becomes accessible to external (authorized) tools and users over the Internet. Tools simply need to provide an interface to access that artifact.

We use the OSLC idea to create an environment of shared model data from all the tools involved in a complex project. This environment, which we call the Link Data and Relations Manager (LDRM), is a repository managed by a server to which all participating tools publish the URIs of their managed resources. The LDRM server builds an index that can reference these resources via their URIs. The LDRM can associate some attributes to the indices. In addition, the LDRM is used to manage the relations between the resources. This ability to model and reference inter-tool relations is a unique and valuable feature, which would be very expensive to implement separately in each tool. The conceptual model of LDRM is illustrated in Figure 1.

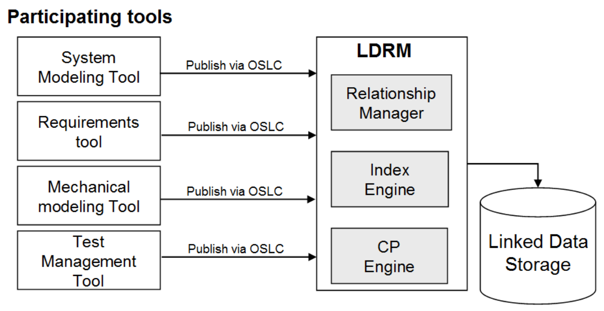

Figure 1: LDRM conceptual model

The LDRM indices and the relations net between them

[1]http://open-services.net

represent the state of the project at a given point in time. Now, it is possible to analyze whether the (possibly partial) relations net complies with the methodology rules, apply corrective measures if needed, refresh the shared model data, and repeat the analysis.

An Illustrative Example Let us demonstrate our proposed approach with a simple example. First, let us consider the following set of methodology rules regarding design artifacts of the types: requirement, system function, and test:

1. Each test covers at least one requirement
2. Each requirement is implemented by a single function
3. Each test is derived from a single function
4. If a test covers a requirement, the test must be derived from a function which is linked to the same requirement.

Such rules may seem arbitrary or even wasteful, but experience shows that obeying these rules saves time and prevents potential errors. For instance, assume that rule 2 is not applied and a certain requirement is implemented by more than one function. If the requirement is changed, as often happens, it is cumbersome and error-prone to detect which functions and interactions between the functions need to be changed in order to accommodate the change in the requirement. Rule 2 ensures that in case of a requirement change, it is clear which is the only function that needs to be changed.

Next, let us consider a toy example of a modern vehicle. The vehicle is a complex system consisting of mechanical, electronic, computer hardware, and software components. At the first stage of the design, the engineers constructed two functions, and three test cases to satisfy three given requirements. Let us further assume that these design artifacts and the relations net between them, which is illustrated in Figure 2, are all held in the LDRM.

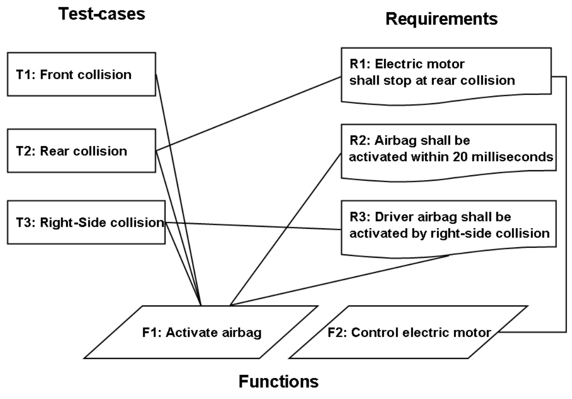
Test-cases
Requirements
T1: Front collision
T2: Rear collision
T3: Right-Side collision
R1: Electric motor shall stop at rear collision
R2: Airbag shall be activated within 20 milliseconds
R3: Driver airbag shall be activated by right-side collision
F1: Activate airbag
F2: Control electric motor
Functions

Figure 2: Simple example of relations net

Now, a rules checker based on the above methodology rules (rules 1-4), which receives the LDRM information will produce an error message indicating that the links between the rear-collision test-case (T2) and the electric-motor requirement (R1) and activate airbag function (F1) are erroneous since they violate the combination of rules 2,3, and 4: the test-case should cover a requirement that is related to the function from which the test is derived. Note that adding a link between F1 and R1, or between T2 and F2 is illegal according to rules 2 and 3, respectively. Hence, the given relations net holds indeed a contradiction.

Next, assume that the engineer corrected this error by canceling the link from T2 to F1. Now, the rules checker will produce a warning saying that according to the rules, the rear-collision test-case T2 must be linked to the control electric motor function F2. Note that this is not an error, since the relations net is not yet final, and links can be added. The rules checker provides further information regarding the linkage between the front-collision test-case (T1) and the requirements, saying that linking this test-case to R1 is forbidden, and that the system engineer should choose at least one of the two other requirements (R2 or R3) to link to this test-case.

2.5 Constraints Programming (CP)

Formally a CP problem is defined as a triple (V, D, C) where V is a set of variables, D their respective finite domains, and C is a set of constraints (or relations) over the variables. A solution to a CP problem is an assignment to each variable out of its domain such that all constraints are satisfied.

The most common algorithms used for solving CP problems involve repeated steps of search and propagation. A variable is defined arc-consistent with another variable if each of its admissible values is consistent with at least one admissible value of the second variable. Algorithms generally use repeating constraint propagations to make all pairs of variables of the problem arc-consistent (Brailsford, Potts, and Smith 1999). Most engines use those "reach arc-consistency" procedures between search steps.

In recent years, some extensions to the formal definition of CP problems were introduced. Among them is the incorporation of preferences and priorities to the problem expressed in the form of soft constraints or an optimization function (Dechter 2003).

An interesting issue in CP is finding the explanation for the unsatisfiability of a problem if no solution can be found. One of the ways of presenting this explanation is called an *unsatisfiable core*. This is a subproblem of the original problem, $UC = (V_U, D, C_U)$, where $V_U \subseteq V$ and $C_U \subseteq C$, which is minimally unsatisfiable. It is minimal in the sense that removing any of the constraints of C_U from UC results in a satisfiable subproblem. Meaning, $(V_U, D, C_U \setminus c)$ is satisfiable for every $c \in C_U$. Note that the domains of the variables in V_U are the same as in the original problem. Of course, an unsatisfiable problem can include several unsatisfiable cores. The one with the least number of constraints is called a minimum unsatisfiable core (Hemery et al. 2006).

CP variables can be defined over various domain types such as integers, floating-point numbers, or strings. In this paper we make use of the slightly more complex domain of set variables. Set variables are variables who's singleton value is a set (in our case, a set of strings). Thus the domain of each set variable is a set of sets. A solution to a CP with set variables includes an assignment of a set to each such variable. During the course of the solution process, domains need to be represented efficiently. Here we choose the bounds-cardinality representation of set-variables domains (Gervet 1997), which consists of three parameters for each set- variable domain: Lower bound (LB), Upper bound (UB), and Cardinality ($card$). Each of these parameters is a set. A domain represented by $LB, UB, card$ contains all the sets S such that $LB \subseteq S \subseteq UB$, and $|S| \in card$.

3 CP Model for Rules Checking

3.1 Translating System Design Data into a Graph

As described above, the system's design data includes the design artifacts and their context, meaning the system's decomposition. We propose describing this data as a double layered graph. The lower layer graph, $G(V_1, E_1)$, represents the relations net between the design artifacts. In this graph, the vertices are the design artifacts and the edges represent the relations between them. In most methodologies, these edges are undirected. The top level graph, $G(V_2, E_2)$ is the system decomposition graph. The vertices represent subsystems and the edges in this graph have the meaning of decomposition. This upper layer graph in most methodologies is a directed acyclic graph with a single source.

The top level graph provides the context information of the design artifacts in the following manner. Each vertex in V_2 acts as a container for all design artifacts (vertices in V_1) that belong to the subsystem it represents. This graphic representation is described schematically in Figure 3.

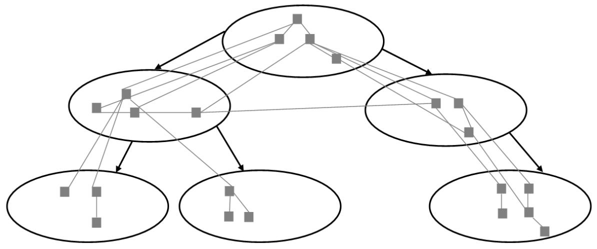

Figure 3: A double layered graph representing the system's data. The small rectangles represent design artifacts V_1, and the large ellipses stand for subsystems V_2.The bold arrows are E_2, and the thin lines are E_1.

These two graphs $G(V_1, E_1)$ and $G(V_2, E_2)$ serve as input for the rules checker along with some additional data on the design artifacts: for each design artifact, we provide its ascription to a subsystem (denoted by i_{sys}), its type (denoted by i_{type}), and perhaps some additional attributes ($i_{attribute}$).

3.2 CP Variables

For each design artifact i, we define a set of CP variables $V_{i,t,s},\ t \in T\ s \in S$. T is the set of artifact types, while S is the set of subsystems (V_2). Each variable is a set variable containing the artifacts of type t in subsystem s linked to artifact i. So, $j \in V_{i,j_{type},j_{sys}}$ means that artifact j is linked to artifact i.

The initial domain of each variable $V_{i,t,s}$ is set by:

$$LB = L \quad (1)$$
$$UB = U \quad (2)$$
$$card = [|L|, |U|] \quad (3)$$

where L is the set of artifacts of type t in subsystem s linked to artifact i in the given relations net, and U is the set of all artifacts of type t in subsystem s. For example, the initial domain of the variable describing T2 links to requirements in our toy example 2.4, $V_{T2,requirement,vehicle}$ is:

$$\begin{aligned} LB &= \{R1\} \\ UB &= \{R1, R2, R3\} \\ card &= \{1, 2, 3\} \end{aligned}$$

3.3 CP Constraints

Analyzing systems engineering methodology rules revealed that all the rules refer to relations between types. Different rules may apply to the linkage between the same types within a subsystem or across subsystems.

We identified three main classes of rules regarding relations between types, taking into account the context information. The first are connectivity rules, which define those types that cannot be linked to each other (eq. 4). The second are cardinality rules, which constrain the number of allowed relations (eq. 5-7), like rules 1,2,3 in the example 2.4. The third class of rules are called cyclic rules since they enforce three-edge cycles in the relations net (see rule 4 in 2.4). The cyclic rules ensure that if two artifacts of predefined types are linked, they both link to the same artifact of a third type (eq. 8). This latter type usually aims to prevent design process errors.

In addition to the design methodology rules, we need to add constraints which ensure that if artifact i is connected to artifact j, then artifact j is connected to artifact i (eq. 9).

$$card(V_{i,B,s}) = 0,\ \forall i : (i_{type} = A\ and\ cond(i_{attribute})) \quad (4)$$
$$card(V_{i,B,s}) = n,\ \forall i : (i_{type} = A\ and\ cond(i_{attribute})) \quad (5)$$
$$card(V_{i,B,s}) \le n,\ \forall i : (i_{type} = A\ and\ cond(i_{attribute})) \quad (6)$$
$$card(V_{i,B,s}) \ge n,\ \forall i : (i_{type} = A\ and\ cond(i_{attribute})) \quad (7)$$
$$(j \in V_{i,B,j_{sys}} \& k \in V_{i,C,k_{sys}}) \Rightarrow k \in V_{j,C,k_{sys}},$$
$$\forall i, j, k : i_{type} = A, j_{type} = B, k_{type} = C \quad (8)$$
$$j \in V_{i,j_{type},j_{sys}} \Leftrightarrow i \in V_{j,i_{type},i_{sys}},\ \forall i, j \quad (9)$$

$cond(i_{attribute})$ stands for an additional optional boolean condition on an attribute associated with artifact i and n stands for a natural number.

$A, B, C \in T$ can be subsets of the types' set and not necessarily denote a single type. The same is true for $s \in S$. If A is a subset, we need to duplicate the constraint for all members of the subset. If B, C or s are subsets, we need to use an auxiliary variable to hold a union between the relevant set variables. For example, for a rule determining that an artifact of type A must be connected to at least one artifact of types B or C, we use an auxiliary variable holding the union of artifacts of type B and C for each subsystem, and define the cardinality constraint on this auxiliary variable. Namely: $card(V_{i,BUC,s}) \geq 1 \Leftrightarrow card(V_{i,B,s} \bigcup V_{i,C,s}) \geq 1$.

Given a specific system data, these methodology rules are automatically translated into specific constraints corresponding to this data. For example, the first methodology rule in our toy example 2.4 is of the form of eq. 7 where $n = 1,\ A = test,\ B = requirement$. Given the design artifacts data, it is translated into the constraints:

$$card(V_{T1,requirement,vehicle}) \geq 1$$
$$card(V_{T2,requirement,vehicle}) \geq 1$$
$$card(V_{T3,requirement,vehicle}) \geq 1$$

The connectivity and cardinality constraints (eq. 4-7) are unary constraints (involve a single variable). As such, they need to be activated only once and hence were included as part of the problem's preprocessing to speed performance. Therefore, some of the violations of the rules could already be detected in this early stage.

3.4 Output Production Algorithms

A solution of the above described CP problem is an extension of the given initial relations net into a net that satisfies all the methodology rules. The extension is done by adding relations. The design artifacts and subsystems themselves cannot be altered using this formulation.

Algorithm 1 CSP application for checking rules

```
Input methodology rules, $G(V_1, E_1)$, $G(V_2, E_2)$,
$\forall i:\ i_{sys}, i_{type}, i_{attribute}$
Output messages to the user
Build CP problem (variables and constraints) from input
Apply connectivity and cardinality rules
if violations exist then
    Produce violations list
    Return
end if
if problem satisfiable then
    Restart problem
    Reach arc-consistency
    Produce information (Algorithm 2)
else
    Extract unsatisfiable core (Algorithm 3)
end if
```

The work flow of the CP application which materializes the methodology support described in Section 2.4, is summarized in Algorithm 1. As explained in section 3.3, some of the violations of local rules can be discovered in the preprocessing stage, before solving the CP formulation. Next, if given the existing relations, there is a solution to the CP problem, the system engineer can proceed in designing the system. To produce a list of warnings and information that may assist the engineer, we rerun the problem only until arc-consistency is reached. We then use the initial and current (after first arc-consistency) domains of the variables to extract this output, as detailed in Algorithm 2 and demonstrated in Table 1.

Algorithm 2 Produce information

```
Input $initial\ LB(V_{i,t,s})$, $initial\ UB(V_{i,t,s})$,
$current\ LB(V_{i,t,s})$, $current\ UB(V_{i,t,s})$,
$current\ card(V_{i,t,s})\ \forall i, t \in T, s \in S$
Output messages to the user
for all $i, t \in T, s \in S$ do
    $implied_{i,t,s} \leftarrow currentLB(V_{i,t,s}) \backslash initialLB(V_{i,t,s})$
    if $implied_{i,t,s} \neq \emptyset$ then
        Produce warning: "item $i$ must link to $implied_{i,t,s}$"
    end if
    $illegal_{i,t,s} \leftarrow initial\ UB(V_{i,t,s})) \setminus current\ UB(V_{i,t,s})$
    if $illegal_{i,t,s} \neq \phi$ then
        Produce info: "item $i$ cannot link to $illegal_{i,t,s}$"
    end if
    $maybe_{i,t,s} \leftarrow current\ UB(V_{i,t,s}) \setminus current\ LB(V_{i,t,s})$
    $n\_maybe_{i,t,s} \leftarrow current\ card(V_{i,t,s}) - |LB(V_{i,t,s})|$
    if $0 \in n\_maybe_{i,t,s}$ then
        Produce info: "item $i$ can link to $(n\_maybe_{i,t,s})$ artifacts from $maybe_{i,t,s}$"
    else
        Produce warning:"item $i$ must link to $(n\_maybe_{i,t,s})$ artifacts from $maybe_{i,t,s}$"
    end if
end for
```

If the CP problem is unsatisfiable, the given relations net contains contradictions. We run an algorithm that finds an unsatisfiable core to identify the part of the relations net and the set of methodology rules that hold the contradiction (Algorithm 3). For our toy example 2.4 and the relation net described in figure 2, the result of Algorithm 3 is:

> Contradiction in: requirement 'Electric motor shall stop at rear collision' must be associated with a single function, and test-case 'Rear collision' must be derived from a single function, and if test case 'Rear collision' and requirement 'Electric motor shall stop at rear collision' are linked, they should link to the same function.

4 Results

The CP application described in section 3 was used to check the rules of a methodology called Embedded Computer-Based System Analysis (ECSAM) (Lavi and Kudish 2005). In the ECSAM methodology, the subsystems structure is a tree and there cannot be relations between artifacts residing in remote subsystems. Hence, there are only three types of viable relations between artifacts with respect to the subsystem they belong to. This allowed us to reduce the set of variables for each design artifact i, to $V_{i,t,s},\ t \in T,\ s \in S$ where T is the set of artifacts types, and $S = in\_same\_system, in\_parent\_system, in\_child\_systems$.

The relations nets we checked belong to a systems engineering tutorial project called Search And Rescue Air Vehicle (SARAH) designed by Israel AeroSpace Industries Ltd. This project includes design artifacts of 7 types residing in

	initial domain	domain after 1st AC	output message
LB	$\emptyset$	$\emptyset$	
UB	R1,R2,R3	R2,R3	info: T1 cannot link to R1
card	0,1,2,3	1,2	warning: T1 must link to 1,2 artifacts from R2,R3

Table 1: Information extracted from the domains of variable $V_{T1,requirement,vehicle}$ by utilizing Algorithm 2

Algorithm 3 Extract unsatisfiable core

```
Input an unsatisfiable problem: $C$ - set of constraints, $V, D$ - set of variables and their initial domains
Output messages to the user
$unsat \leftarrow \{\}$ ($unsat$ - set of constraints that comprise an unsatisfiable core ($C_U$))
$notInUnsat \leftarrow \{\}$
for all $c \in C$ do
    if $(V, D, C/c)$ satisfiable then
        $unsat \leftarrow unsat \cup c$
        if (V,D,$unsat$) unsatisfiable then
            return
        end if
    else
        $notInUnsat \leftarrow notInUnsat \cup c$
        $C \leftarrow C/c$
    end if
end for
Produce contradictions list from $(V, D, unsat)$
```

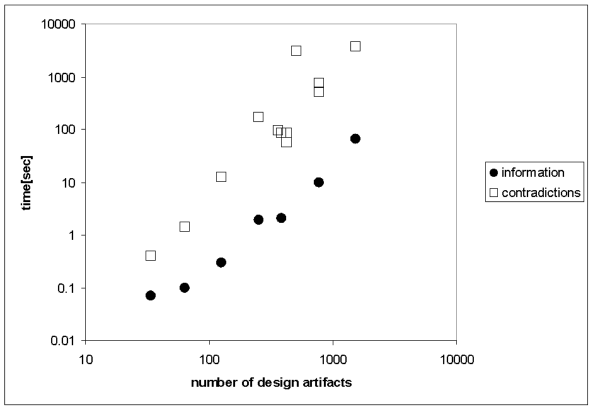

Figure 4: Total runtime (in seconds) for different instances. The rectangles represent instances requiring contradiction analysis (Algorithm 3), and the circles represent instances in which information was produced (Algorithm 2).

up to 31 subsystems. The context tree has a depth of 4. We have enforced 76 methodology rules on this system's data.

Our rules checker operated on different instances (given relations nets). The size of the instances determined the size of the generated CP problems which ranged from 750 to 33,794 variables and from 115 to 10,600 constraints. We solved these generated CP problems using an IBM CP solver called GenerationCore [2] running on an Intel 2.4GHz processor with 2.4GB memory.

The performance results of the application are summarized in figure 4 for the different output types: information production utilizing Algorithm 2, or contradiction analysis utilizing Algorithm 3.

5 Discussion

The CP application described is unique in the sense that it must determine if a certain instance is satisfiable, but there is no use for the explicit solution. This may change if preferences are added to the model in the form of soft constraints or an optimization function; in this case the solution could be used to direct the engineer towards a better design. As shown earlier, the CP application can also give indications to the engineer whether links are implied, optional, or forbidden, and hence provide decision support. If preferences are considered, this decision support can be greatly enhanced.

Modeling the rule checking problem through set variables, and using the representation of the set that has a cardinality ingredient proved very helpful. We found that in order to ease the search steps for finding the problem's solution, it is better to use many variables with smaller domains, than a small number of variables with larger domains. Therefore, we recommend using many types of design artifacts. This contributes to the precision of the methodology rules and improves the application performance.

The novel representation of the system design data as a two layers graph, not only facilitated formulation of the methodology rules, but also proved to be useful for other uses of the LDRM. Since this graph contributes for managing and presenting large amounts of design artifacts and relations in a user friendly way, it eases traceability and allows for a comprehensive impact analysis.

The fact that the rules checker can point out contradictions in a partially linked system model and not only when the model is fully linked was found very practical in finding errors in the design process. Therefore, providing explanations in case of unsatisfiability is a crucial feature of the application. However, the straightforward algorithm we use for providing these explanations can be insufficient in some cases. This algorithm finds only a single unsatisfiable core at a time and not necessarily the minimum one but the one whose first member appears last in the constraints order. In addition, the CP formulation assumes the existence of all of relevant design artifacts, hence its information and explanations regarding a contradiction source may be ill-phrased. For example: let us look at a methodology that dictates that

[2]http://www.research.ibm.com/haifa/dept/vst/csp_gec.shtml

any artifact of type A is linked to exactly one artifact of type B and vice versa. In a system having three artifacts of type A and two artifacts of type B, the application would indeed indicate a contradiction. But, it would point out its source as the five constraints regarding the five existing artifacts, while a more concise explanation is a missing artifact of type B.

Figure 4 demonstrates that the application performance for a compliant relations net (where information is produced) is much better than for non compliant relations net (where contradictions are analyzed) of the same size. Therefore, we suspect that we would encounter a scalability problem when dealing with large non-compliant relations net. It should be noted that determining whether the relations net is in compliance with the rules and providing a violations list (if exist) is almost immediate. It is the process of analyzing contradictions by extracting an unsatisfiable core which is time-consuming, since many subproblems can be solved until an answer is reached. One way to overcome this obstacle is through smart ordering of variables and constraints, such that those most likely to participate in an unsatisfiable core will be checked first. Another way is by setting a timeout for the application and printing a partial unsatisfiable core.

Figure 4 also demonstrates that the application's runtime (for both possible outputs) is roughly polynomial in the number of design artifacts of the given relations net. The tutorial engineering project we used to test the application is equivalent to a small to medium sized real project. For real-life projects, running the application over-night would be sufficient. Hence, we believe the application performance may be suitable for real projects.

In companies using ECSAM methodology and having a partially automated rules checking procedure, the application is estimated to save 10 hours of the engineers' workload per subsystem. For companies where the process is completely manual, the workload savings are even larger. Yet, the largest benefit of the application is increasing the number of detected errors during the design phase.

The use of CP technology provides a declarative language for describing the rules. This results in a much simpler way to define the rules and to understand them. It also saves the effort needed to write scripts that check data compliance to the rules. Hence, checking even complicated rules is easily enabled. A large variety of additional types of rules can be supported by CP technology. The inherent flexibility in the rules definition allows us also to easily test and update the methodology itself.

It is worth mentioning that most companies use the same methodology for all projects. In such companies, the methodology rules should be formulated as CP constraints only once, and the application will automatically generate for every design the corresponding CP problems for checking the design conformance to the methodology rules. To this end, a way to introduce the design artifacts data and the relations net between them to the application is needed. Note that the described application can also be used to check only part of the methodology rules and can be applied within any tool which holds a (partial) relations net, not necessarily the LDRM. If our formulation of the CP problem is to be used, a CP solver supporting set variables is also needed.

6 Summary and Conclusions

We make use of a newly proposed framework for designing complex systems. This framework involves a central repository called LDRM, which holds indexed references to the design artifacts residing in the design tools, and the relations between those artifacts.

Using a CP-based application and a novel representation of the system's data combined with the LDRM capabilities, allowed us to simultaneously check a set of possibly complex methodology rules involving design artifacts from several different tools. We show that the application can check the conformance to complex global rules as well as simple local ones. Using CP technology in a novel way allows the rules checker to alert the system engineer in case of violations or contradictions even when given a partial relations net between the design artifacts. It can also provide the system engineer with information that may assist in the subsequent stages of the design process. These capabilities can help improve productivity and enforce process methodology.

The CP application also offers a means for the process engineer to easily define, alter, and check the methodology thanks to the flexibility in the rules definition.

There is room for improvement in the performance and scalability of the application, especially in the case of contradictions. This improvement can be achieved by using better algorithms for extracting unsatisfiable cores.

Acknowledgments. We thank Hisashi Miyashita, Hiroaki Nakamura, Kohei Suenaga, Futoshi Iwama and Kohichi Kamijoh from the Model Driven Systems Engineering group in IBM Research-Tokyo for initiating this research and for contributing to the LDRM realization.

References

Brailsford, S.; Potts, C.; and Smith, B. 1999. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research* 119(3):557–581.

Dechter, R. 2003. *Constraint Processing*. Elsevier.

Engel, A. 2010. *Verification, Validation and Testing of Engineered Systems*. John Wiley & Sons.

Gervet, C. 1997. Interval propagation to reason about sets: definition and implementation of a practical language. *Constraints* 1(3):191–244.

Gery, E.; Modai, A.; and Mashkif, N. 2010. Building a smarter systems engineering environment. In *Innovate'10*.

Hemery, F.; Lecoutre, C.; Sais, L.; and Boussemart, F. 2006. Extracting mucs from constraint network. In Brewka, G., ed., *ECAI'06, Proceedings of the 17th European Conference on Artificial Intelligence*, 113–117.

Holt, J., and Perry, S. 2008. *SysML for Systems Engineering*. The Institution of Engineering and Technology, London, UK.

Lavi, J. Z., and Kudish, J. 2005. *Systems Modeling & Requirements Specification Using ECSAM: A Method for Embedded Computer-Based Systems Analysis*. Dorset House.

Steinberg, D.; Budinsky, F.; Paternostro, M.; and Merks, E. 2008. *EMF: Eclipse Modeling Framework*. Addison-Wesly Professional.