Coordination specification of the analytical target cascading process using the Chi language

Published in:
9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, Georgia, September 4-6, 2002

Published: 01/01/2002

Document Version
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain.
- You may freely distribute the URL identifying the publication in the public portal.

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 02. Feb. 2018
COORDINATION SPECIFICATION OF THE ANALYTICAL TARGET CASCADING PROCESS USING THE χ LANGUAGE

L.F.P. Etman
Eindhoven University of Technology, Eindhoven, The Netherlands

and

M. Kokkolaras and P.Y. Papalambros
University of Michigan, Ann Arbor, Michigan

and

A.T. HofKamp and J.E. Rooda
Eindhoven University of Technology, Eindhoven, The Netherlands

ABSTRACT

Optimal design problems of large-scale and complex engineering systems are typically decomposed into a number of smaller and tractable subproblems. Analytical target cascading (ATC) is a methodology for translating overall system design targets to individual specifications for the subsystems and components that make up the system based on a hierarchical partition. We propose to use the χ language and software tools to specify and implement the coordination of the analytical target cascading process. ATC is implemented as parallel processes that exchange data via channels, which represent the links between the subproblems. The process specifications define how individual processes communicate with other coupled processes. We show the advantages of χ for coordinating the ATC process by means of an illustrative example, and demonstrate that different coordination strategies can be implemented and evaluated efficiently.

1 INTRODUCTION

A key challenge in early product development stages of complex products is to propagate desirable product characteristics, defined by product’s specifications (or targets), to the various subsystems and components in a consistent and efficient manner. Consistency means that all parts of the designed system should end up working well together, while efficiency means that iterations at later product development stages, which are costly in time and resources, should be avoided.

Analytical target cascading is a methodology for the design of large engineering systems at the early product development stages (Kim, 2001). First, the design problem is partitioned into a hierarchical set of subproblems associated with systems, subsystems, and components. Design specifications (or targets) are defined at the top level of the multilevel design formulation and “cascaded down” to lower levels. Design subproblems are formulated at each level so that components, subsystems, and systems are designed to match the cascaded targets consistent with the overall system targets. The main benefits of target cascading are reduction in design-cycle time, avoidance of design iterations late in the development process, and increased likelihood that physical prototypes will be closer to production quality. Target cascading also facilitates concurrency in system design, allowing the outsourcing of subsystems and components to suppliers. Target cascading offers a robust framework for multilevel design and has been demonstrated (Michelena et al., 2002) to be convergent under standard convexity and smoothness assumptions, whereas other similar problem formulations exhibit convergence difficulties.

The ATC process has been applied successfully to a wide range of vehicle design applications (Kim et al., 2000, Kim et al., 2001, Michelena et al., 2001, Kim et al., 2002). It has also been extended to the design of product families (Kokkolaras et al., 2002). The key to the success of the ATC process lies in coordinating the solution process of the subproblems. Several coordination strategies are presented in (Michelena et al., 2002). They recursively solve one subset of two coupled subproblems at a time. Although they do not address this, their strategies allow the concurrent solution of subproblems (e.g., at the same level of the hierarchy). For larger decompositions the implementation of the ATC coordination may become complicated and prone to errors, even if one refrains from the parallel solutions of the subproblems.

A more precise description of the ATC coordination, especially in regard to concurrency, would be advantageous. Etman et al. (2002) suggest to use a programming language founded on concurrency theory for this purpose. They adopt the χ specification lan-
language (Kleijn and Roorda, 2001; Roorda, 2000) that is based on concepts of Communicating Sequential Processes (CSP) (Hoare, 1985). They show that such a language enables to model and implement the coordination of coupled subproblems in multidisciplinary optimization (MDO) including the possibly parallel behavior of the subproblems.

This paper proposes the use of $\chi$ to specify and implement the coordination process of ATC. Our intention is to demonstrate the advantages of $\chi$ by evaluating the performance of alternative coordination strategies for a multilevel target cascading problem. The paper is organized as follows. The mathematical formulation of the analytical target cascading process is given in the next section. The coordination specification of ATC is put into the $\chi$ perspective in Section 3. An illustrative example is presented in Section 4, where different coordination strategies are implemented by means of $\chi$ and evaluated. Conclusions are drawn in Section 5.

2 TARGET CASCADING FORMULATION

The analytical target cascading process is presented using a general notation, from which the design problem for each element (i.e., system, subsystem, or component) can be recovered as a special case. The formulation allows for design specifications to be introduced not only at the top level for the overall product, but also “locally” to account for individual system, subsystem, and/or component requirements. To represent the hierarchy of the partitioned design problem, the set $E_i$ is defined at each level $i$, in which all the elements of the level are included. For each element $j$ in the set $E_i$, the set of children $C_j$ is defined, which includes the elements of the set $E_{i+1}$ that are children of the element. An illustrative example is presented on Figure 1: At level $i = 1$ of the partitioned problem we have $E_1 = \{B, C\}$, and for element “B” on that level we have $C_B = \{D, E\}$. There are two types of responses:

- responses $R$ linked to “local” targets (e.g., at the top level), and responses $R$ linked to “cascaded” targets, i.e., linking two successive levels in the problem hierarchy. The design problem $P_{ij}$ corresponding to the $j$-th element at the $i$-th level is formulated as follows:

$$
\min_{x_{ij}} \|\hat{R}_{ij} - T_{ij}\| + \|R_{ij} - R_{ij}^L\| + \|y_{ij} - y_{ij}^U\| + e_{ij}^R + e_{ij}^t
$$

subject to

$$
\sum_{k \in C_j} \|\hat{R}_{ij} - \hat{R}_{ij}^L\| \leq e_{ij}^R
$$

$$
\sum_{k \in C_j} \|y_{ij} - y_{ij}^L\| \leq e_{ij}^t
$$

where $\hat{R}_{ij} = [\hat{R}_{ij}^L, \hat{R}_{ij}^U]^T = r_{ij}(\hat{R}_{i+1}^L, \ldots, \hat{R}_{i+1}^{L_{k_{ij}}}, x_{ij}, y_{ij}), C_j = \{k_1, \ldots, k_{c_{ij}}\}$, and $c_{ij}$ is the number of child elements. Note that an element’s response depends on the element’s design variables as well as on its children’s responses. In the above problem formulation,

- $\hat{x}_{ij} = [x_{ij}^L, y_{ij}^L, y_{ij}^U, \ldots, y_{ij}^{L_{k_{ij}}}, \hat{R}_{i+1}^L, \hat{R}_{i+1}^U, e_{ij}^R, e_{ij}^t]^T$ is the vector of all optimization variables,

- $\hat{x}_{ij}$ is the vector of local design variables, that is, variables exclusively associated with the element,

- $y_{ij}$ is the vector of linking design variables, that is, variables associated with two or more elements that share the same parent,

- $\hat{R}_{ij}$ is the vector of all responses,

- $e_{ij}^R$ is the tolerance optimization variable for coordinating the responses of the children of the element,

- $e_{ij}^t$ is the tolerance optimization variable for coordinating the linking design variables of the children of the element,

- $T_{ij}$ is the vector of local target values,

- $R_{ij}^L$ is the vector of response values cascaded down to the element from its parent,

- $y_{ij}^L$ is the vector of linking design variable values cascaded down to the element from its parent,

- $\hat{R}_{i+1}^L$ is the vector of response values cascaded up to the element from its $k$-th child,

- $y_{ij}^{L_{k_{ij}}}$ is the vector of linking design variable values cascaded up to the element from its $k$-th child,

- $g_{ij}$ and $h_{ij}$ are vector functions representing inequality and equality design constraints, respectively.

2 American Institute of Aeronautics and Astronautics
3 ATC COORDINATION SPECIFICATION

Analytical target cascading requires the iterative solution of the optimization subproblems according to the hierarchical structure of the decomposed problem. In this iterative coordination process the subproblem targets and design variables need to be found such that they are consistent with respect to the top level objectives and the coupling constraints of the subproblems.

Several different iterative coordination strategies may lead a consistent system design. Michelena et al. (2002) proved that a class of ATC coordination strategies are guaranteed to converge under standard smoothness and convexity assumptions. However, it is not known a priori which strategy is the most efficient. This may even be problem dependent. In this respect one may also want to distinguish between the number of subproblems that can be solved in parallel. It would be advantageous to have a coordination language with a clear concept of concurrency that enables rapid implementation and investigation of alternative ATC coordination strategies. The purpose of the present paper is to demonstrate this by means of the specification language $\chi$ (see Etman et al. (2002)).

This section explains how the ATC coordination can be specified as a coupled system of $\chi$ processes. A process will be instantiated for each design problem $P_{ij}$. This process specifies the sequence of statements to exchange data with its parents and children, and to carry out a numerical optimization of subproblem $P_{ij}$. In the general form of Section 2, such an ATC subproblem process receives target and constraint values from its parent and children, respectively, and returns updates after the associated optimization problem has been solved. The exact process specification for problem $P_{ij}$ depends on the coordination scheme that is employed to solve the overall ATC problem.

3.1 Downwards-only sequential coordination

Consider first a coordination scheme that solves optimization subproblems iteratively in a downwards level-by-level sequence. That is, optimization subproblems are solved subsequently in increasing level order: $i = 0, 1, \ldots, N, 0, \ldots, N$ etc., where $i = 0$ and $i = N$ represent top and bottom level, respectively.

In the downwards-only coordination, the top-level process $C_{\text{top}}$ for problem $P_0$ (index $j$ is dropped at the top level since there is only one element) starts by carrying out an optimization to determine response and linking variable targets that will be cascaded to its children. At the first iteration, $C_{\text{top}}$ uses some user-provided initial guess of the response and linking variable values regularly passed up from its children. After $C_{\text{top}}$ has cascaded its targets, it waits until it receives updated values of children response and linking variables. It then carries out a new optimization, compares the new vector of optimization variables to the previous one, examines convergence of the children processes, and either converges or reiterates.

A second type of process is needed for the intermediate-level subproblems $P_{ij}$ that have both a parent and one or more children, i.e., $i \in 1, \ldots, N - 1$. For the downwards-only scheme, such an intermediate-level process $C_{\text{mid}}$ carries out the following sequence of steps: Receive targets from the parent; carry out an optimization (at the first iteration use an initial guess for the children response and linking values); cascade determined targets to the children; wait until all the children have sent updated response and linking values; and finally pass updated values to the parent.

Thirdly, we need bottom-level processes $C_{\text{bot}}$ for the bottom-level subproblems $P_{Nj}$. A bottom-level process waits until it receives targets from its parent, carries out an optimization, and returns updated values.

The downwards-only coordination scheme for a decomposed problem as depicted in Figure 1 is specified simply by coupling instances of top-level, intermediate-level, and bottom-level processes. No additional process is needed to control the overall coordination. Initially, all processes are waiting to receive target data from their parents, except for the top-level process. The top-level process carries out an optimization and cascades targets down to its children. The intermediate-level processes may carry out their optimizations in parallel; all the other processes are waiting to receive new data. The computed targets are cascaded down. This proceeds until the bottom-level processes have been reached and have carried out their optimizations. The bottom-level processes pass updated response and linking target values to their parents, which update their parents without carrying out any new optimizations. In this manner, updated target values are rebalanced up level-by-level until the top-level process is finally reached. If convergence has not occurred, a new ATC iteration starts.

Note that the tree of processes does not need to be symmetrical. The same coordination arises if one branch has more levels than another branch. In case the ATC partition consists of just two levels, the coordination specification contains only top-level and bottom-level process instances.

Finally, a repository process $R$ is introduced to facilitate the monitoring of all processes during the ATC iterations. The subproblem processes send updated values of their optimization variables to this process $R$. These optimization variable values will be stored by $R$. After completion of the ATC, the complete iteration history is available from $R$ and may be used for further analysis.
3.2 Definition of $\chi$ processes

The $\chi$ specifications of the processes in the downwards-only sequential coordination scheme are presented below. The following variable types are introduced:

$$\text{type } \chi = \text{real}^{ms}$$

$$\chi \chi \chi \chi \chi \chi \chi \chi \chi$$

$$\chi \chi \chi \chi \chi \chi \chi \chi \chi$$

$$\chi \chi \chi \chi \chi \chi \chi \chi \chi$$

Herein, types $\chi$, $\chi$, and $\chi$ denote vector arrays of reals of fixed (maximum) length $ms$, $my$, and $mr$, respectively; $\chi$ is a strongly typed language requiring that the dimensions of arrays are known at compilation time (Etman et al., 2002). Types $\chi v s$ and $\chi v s$ are matrix arrays of sizes $ms \times mr$ and $ms \times my$, respectively. Types $\chi v b t o p$, $\chi v b m i d$, and $\chi v b b o t$ are tuples containing two or more data-elements of type $\chi v s$, $\chi v y$, $\chi v y s$, $\chi v r s$, or real. These three tuple types match the optimization variables of the top-level, intermediate-level, and bottom-level design problems, respectively. The elements in the tuples can be accessed through their identifiers defined before the dot operator. Finally, type $\chi$ is defined as a tuple of reals and nats to store the scaling values used in the optimization problem of the process, the number of children of the process, and the actual array dimensions used in the process.

Using this type definitions, the $\chi$ specification of the top-level process $C_{\chi \chi \chi \chi}$ is:

$$\text{proc } C_{\chi \chi \chi \chi}(f : \chi v b t o p, \chi v r s, \chi v y s, \chi) \rightarrow \chi v b t o p \times \chi v r s$$

The first element $f$ of the parameter list of $C_{\chi \chi \chi \chi}$ defines the Matlab function that will carry out the optimization. Parameters $b$ and $c$ represent the send and receive port arrays, respectively, through which data is sent to and received from the children of $C_{\chi \chi \chi \chi}$. The fourth parameter $e$ is a send port to repository $R$ to update the coordination history; the fifth parameter $s$ is a synchronization port to $R$ to notify when the ATC coordination has been finished. The last four parameters are the top-level response targets, the initial values of the optimization variables, the subproblem parameter values, and the convergence tolerance, respectively.

The following local variables are introduced in $C_{\chi \chi \chi \chi}$: a response vector $r$, matrices $R_s$ and $Y_s$ of response target values and linking variable target values, respectively, that have been passed up from the children, a tuple of optimization variables $x_b$, a boolean variable $c v r g$, a boolean array $c v r g s$, and two natural variables $i$ and $n$. $C_{\chi \chi \chi \chi}$ starts by carrying out an initial, one-time only optimization. The following sequence of tasks is then repeated: cascading targets, receiving updated
target values, carrying out optimization, checking convergence, and updating optimization variables.

$C_{top}$ is declared locally converged if the square of the norm of the difference between previous and current iterates is smaller than some predefined value $tol$, i.e., if $||x^{(n)} - x^{(n-1)}||_2^2 \leq tol$. Note that the vector of optimization variables $x$ has different instantiations for the top, intermediate, and bottom levels. $C_{top}$ is declared globally converged if convergence has occurred for the "local" optimization problem as well as for all the children optimization problems. The function $checkcvrg$ returns true if this is the case. To this end, the children pass up their convergence status in addition to the updated response and linking variable values. $C_{top}$ stores the convergence status of its children in the boolean array $cvrgs$. The functions $topnorm$ and $checkcvrg$ are specified as $y$ functions. The ATC process is terminated when $C_{top}$ has converged or the maximum number of iterations has been reached. A synchronization is sent to repository process $R$ to acknowledge this.

The intermediate-level process $C_{mid}$ is specified in a similar way:

$$\text{proc } C_{mid}(f : (vxbmid, vr, vy, vrs, vys, par) \rightarrow vxbmid \times vr$$
$$\quad, a : ?vr \times vy$$
$$\quad, b : (?vr \times vy)^{ms}$$
$$\quad, c : (?vr \times vy \times bool)^{ms}$$
$$\quad, d : ?vr \times vy \times bool$$
$$\quad, e : !vxbmid, x\theta : vxbmid, p : par$$
$$\quad, tol : real) =$$
$$\begin{array}{l}
[ r, rUS : vr, rLs : vrs, yUS : vy, yLs : vys$
\quad, xb : vxbmid, cvrg : bool, cvrgs : bool^{ms}, i : nat$
\quad, rLs := xbo.0.r; yLs := xbo.0.y;]
\quad,*[true$
\quad\rightarrow a?(rUS, yUS)$
\quad\quad; \langle x, y \rangle := f(xbo, rUS, yUS, rLs, yLs, p)$
\quad\quad; e!xb$
\quad\quad; i := 0$
\quad\quad; *[i < p. ns$
\quad\quad\quad\rightarrow b!((xb.rs).i, (xb ys).i)$
\quad\quad\quad; i := i + 1$\quad\quad\]$\quad\quad; i := 0$
\quad\quad; *[i < p. ns$
\quad\quad\quad\rightarrow c.i?(rLs.i, yLs.i, cvrgs.i)$
\quad\quad\quad; i := i + 1$\quad\quad\]$\quad; cvrg := midnorm(xbo, xb, p) < tol$
\quad; cvrg := checkcvrg(cvrg, cvrgs, p.ns)$
\quad; xbo := xb$
\quad; d!(r, xb, y, cvrg)$
\quad; ]
\end{array}$$

$C_{mid}$ has four communication ports: $a$ and $d$ are receive and sent ports coupled to the parent; $b$ and $c$ are receive and sent port arrays (bundles), the elements of which are coupled to the children through channels. $C_{mid}$ receives response and linking variable targets through port $a$; cascade targets to its children through ports $b; i$; receives updated response and linking variable values as well as the status of convergence of children-problems through ports $c;i$; and finally passes updated values and convergence status to its parent through port $d$. This sequence is repeated indefinitely. After $C_{mid}$ has received the target values cascaded down from its parent, it carries out the local optimization defined by function $f$, using initial optimization variable values $xbo$ and subsystem parameters $p$. After the convergence status of the children has been received, the local and overall convergence of $C_{mid}$ is determined as described for $C_{top}$; however, a modified norm function ($midnorm$) is used.

The bottom-level process $C_{bot}$ is specified next. $C_{bot}$ receives targets through port $a$, carries out the optimization defined by $f$, and passes $up$ updated values and convergence status through port $d$:

$$\text{proc } C_{bot}(f : (vxbot, vr, vy, par) \rightarrow vxbot \times vr$$
$$\quad, a : ?vr \times vy, d : !vr \times vy \times bool$$
$$\quad, e : !vxbot$$
$$\quad, xbo : vxbot, p : par, tol : real) =$$
$$\begin{array}{l}
[ r, rUS : vr, yUS : vy, xb : vxbot, cvrg : bool$
\quad, *[true$
\quad\rightarrow a?(rUS, yUS)$
\quad\quad; \langle x, y \rangle := f(xbo, rUS, yUS, p)$
\quad\quad; e!xb$
\quad; cvrg := botnorm(xbo, xb, p) < tol$
\quad; xbo := xb$
\quad; d!(r, xb, y, cvrg)$
\quad; ]
\end{array}$$

Finally, there is a repository process $R$ that collects optimization results every time a problem is solved, and stores them in lists:

American Institute of Aeronautics and Astronautics
The repetitive selective waiting statement waits until a new iteration update is received from one of the subproblem processes or until a synchronization is received from the top-level process for the acknowledgement of the ATC finish. The key statement of $R$ is a repetitive coordination scheme. For example, the nested coordination that corresponds to the scheme depicted in Figure 2, is obtained by simply inserting an iteration loop between the receive and send statements with respect to the parent-related communication. In this manner, communications with the children are nested with respect to communications with the parent. The repetition statement is inserted into (7.3) as follows.

```
;true; ?vt
→ td := td + [vt]
[ j : nat ← 0...nm : true; m.j?vm
→ md. j := md. j + + [vm]
[ j : nat ← 0...nb : true; b.j?vb
→ bd. j := bd. j + + [vb]
true; ?s
→ b := pp(td, md, bd, tol, sca)
]
]
```

Figure 2: Top-down nested coordination.

3.3 Alternative coordination schemes

Additional coordination schemes can be obtained by modifying slightly the specifications of the subproblem processes. For example, a downwards-upwards sequential coordination scheme ($i = 0, 1, \ldots, N - 1, N, N - 1, \ldots, 1, 0,$ etc.) is obtained by simply inserting the line

```
; (xb, r) := f(xb0, rUS, yUS, rLs, yLs, p)
```

after the statement

```
; [i < p.ns
→ c.i? (rLs, i, yLs, i, cvrsgs, i)
; i := i + 1
]
```

in the $C_{mid}$ process specification (7.3). By doing this, intermediate-level processes carry out an additional optimization every time updated values are rebalanced up.

Michelena et al. (2002) consider nested coordination schemes. For example, the nested coordination that corresponds to the scheme depicted in Figure 2, is

```
;[true; a? (rUS, yUS)
; ] [cvrg ∧ n < maxiter
→ n := n + 1
; (xb, r) :=
→ f(xb0, rUS, yUS, rLs, yLs, p)
; d! (r, xb, y, cvrsg)
]
```

4 Example

We will now demonstrate the implementation of different coordination strategies using the $\chi$ language on a simple but illustrative analytical optimization problem.
4.1 Mathematical formulation of example problem

Our example is based on the geometric programming problem presented in Kim (2001):

\[
\begin{align*}
\min_{z \geq 0} & \quad z_1^2 + z_2^2 \\
\text{s.t.} & \quad \frac{z_3^2 + z_4^2}{z_3^2} - 1 \leq 0 ; \quad \frac{z_5^2 + z_6^2}{z_5^2} - 1 \leq 0 \\
& \quad \frac{z_7^2 + z_9^2}{z_7^2} - 1 \leq 0 ; \quad \frac{z_8^2 + z_{10}^2}{z_8^2} - 1 \leq 0 \\
& \quad \frac{z_{11}^2 + z_{12}^2}{z_{11}^2} - 1 \leq 0 ; \quad \frac{z_{13}^2 + z_{14}^2}{z_{13}^2} - 1 \leq 0 \\
& \quad z_1 - z_2 - z_3^2 - z_4^2 - z_5^2 - z_6^2 - z_7^2 - z_8^2 - z_{10}^2 - z_{11}^2 - z_{12}^2 - z_{13}^2 - z_{14}^2 = 0
\end{align*}
\]

The solution of problem (2) is known to be equal to \( z = [2.84, 3.09, 2.36, 0.76, 0.87, 2.81, 0.94, 0.97, 0.87, 0.80, 1.30, 0.84, 1.76, 1.55] \). Kim (2001) decomposed this problem using the equality constraints as responses within a bi-level hierarchical structure and demonstrated the application of the ATC process. Here, we decompose the original problem into three levels as shown in Figure 3; \( z_5 \) is the linking variable that couples the subproblems of the intermediate level. Note that \( z_{11} \) is a linking variable coupling the two problems of the bottom level. The ATC formulation does not allow subproblems to share variables unless they are children of the same parent. Since the purpose of this example is to illustrate the \( \chi \) implementation of alternative coordination strategies, we treat \( z_{11} \) as a parameter using its known optimal value.

The subproblems are formulated in the next subsections following the notation presented in Section 2; the index \( j \) is dropped for the top-level problem since there is only one element.

Figure 3: Hierarchical structure of decomposed problem

---

Top-level problem

\[
\begin{align*}
\min_{R_{11}, R_{12}, \eta_0} & \quad \|R_{11} - R_0\| + \|y_0 - y_{11}\| + \|y_0 - y_{12}\| + \|y_0 - R_{11}\| + \|y_0 - R_{12}\| \\
\text{subject to} & \quad \|y_0 - y_{11}\| + \|y_0 - y_{12}\| \leq \eta_0 \;
\end{align*}
\]

Intermediate-level problems

There are two problems at the intermediate level. Problem \( P_{11} \) is formulated as

\[
\begin{align*}
\min_{R_{11}, R_{12}, \eta_1} & \quad \|R_{11} - R_{11}\| + \|y_{11} - y_{11}\| + \|y_{11} - y_{12}\| + \|y_{11} - R_{11}\| + \|y_{11} - R_{12}\| \\
\text{subject to} & \quad \|R_{11} - R_{11}\| + \|y_{11} - y_{11}\| \leq \eta_1, \;
\end{align*}
\]

where \( R_{11} := z_1, \; R_{12} := z_2, \; y_{11} := z_5, \; R_0 = \eta_0(R_{11}, R_{12}) = \frac{z_7^2 + z_8^2}{z_7^2} + \frac{z_9^2 + z_{10}^2}{z_9^2}, \; \text{and} \; y_{12} := z_5, \; R_{11} = \sqrt{\frac{z_3^2 + z_4^2 + z_5^2}{z_3^2} + \frac{z_{11}^2 + z_{12}^2 + z_{13}^2 + z_{14}^2}{z_{11}^2}}.

Bottom-level problems

There are two problems at the bottom level. Problem \( P_{21} \) is given by

\[
\begin{align*}
\min_{x_{21}} & \quad \|R_{21} - R_{21}\| \\
\text{subject to} & \quad g_{21}(x_{21}) \leq 0
\end{align*}
\]

where \( x_{21} := [z_6, z_9, z_{10}], \; p = 1.3(= z_{11}), \; R_{21} = r_{21}(x_{21}) = \sqrt{\frac{z_6^2 + z_9^2 + z_{10}^2}{z_6^2} + p^2}, \; \text{and} \;
\begin{align*}
g_{21}(x_{21}) &= \left[ \frac{(z_6^2 + z_9^2 + z_{10}^2)^{p-2} - 1}{(z_6^2 + z_9^2 + z_{10}^2)^{p-2} - 1} \right].
\end{align*}

The formulation of problem \( P_{22} \) is

\[
\begin{align*}
\min_{x_{22}} & \quad \|R_{22} - R_{22}\| \\
\text{subject to} & \quad g_{22}(x_{22}) \leq 0
\end{align*}
\]

where \( x_{22} := [z_6, z_9, z_{10}], \; p = 1.3(= z_{11}), \; R_{22} = r_{22}(x_{21}) = \sqrt{\frac{z_6^2 + z_9^2 + z_{10}^2}{z_6^2} + p^2}, \; \text{and} \;
\begin{align*}
g_{22}(x_{22}) &= \left[ \frac{(z_6^2 + z_9^2 + z_{10}^2)^{p-2} - 1}{(z_6^2 + z_9^2 + z_{10}^2)^{p-2} - 1} \right].
\end{align*}


where \(x_{22} := [z_{12}, z_{13}, z_{14}]\), \(p = 1.3(= z_{11})\), \(R_{22} = r_{22}(x_{22}) = \sqrt{z_{12}^2 + z_{13}^2 + z_{14}^2},\) and
\[
g_{22}(x_{22}) = \begin{bmatrix} (p^2 + z_{12}^2 - 1) \\ (p^2 + z_{12}^2 - 1) \end{bmatrix}.
\]

4.2 Implementation of coordination using \(\chi\)

The \(\chi\) implementation of the three-level ATC example requires a number of processes as specified in Section 3. We assign the following constant values first:

\[
\begin{align*}
\text{const} & : \text{nat} = 3 \\
mx & : \text{nat} = 1 \\
mb & : \text{nat} = 2 \\
mn & : \text{nat} = 2 \\
nb & : \text{nat} = 2 \\
maxiter & : \text{nat} = 1000 \\
nan & : \text{real} = -9.9e99
\end{align*}
\]

where \(mx\) is the maximum length of design variable arrays (there are maximum three local design variables in any of the subproblems), \(mb\) is the maximum length of response variable arrays, \(mn\) is the maximum length of linking variable arrays, \(nb\) is the maximum number of children in any subproblem (2 in the top-level problem), \(mn\) is the number of intermediate-level processes \((P_{11} \text{ and } P_{12})\), and \(nb\) is the number of bottom-level processes \((P_{21} \text{ and } P_{22})\). Moreover, we define the maximum number of iterations \(maxiter\) and a “not-a-number” value \(nan\) that is used for empty entries in the fixed-length arrays.

The system that couples the processes is instantiated according to Figure 4. The top-level problem

\[
P_0 \text{ requires a process instantiation of } C_{\text{top}}. \text{ Each of the two children of } P_0 \text{ requires an instantiation of the intermediate-level process } C_{\text{mid}}, \text{ representing problems } P_{11} \text{ and } P_{12}. \text{ The bottom-level subproblems } P_{21} \text{ and } P_{22}
\]

are represented by two process instantiations of \(C_{\text{bot}}\). Finally, repository process \(R\) needs to be instantiated.

These six processes have to be coupled to each other by channels of appropriate data type. The channel names are defined in Figure 4. For example, \(C_{\text{top}}\) sends data of type \(vr \times vy\) to each of its children via channel array \(a\), and receives data of type \(vr \times vy \times \text{bool}\) from its children via channel array \(b\). \(C_{\text{top}}\) shares two additional synchronization send channels with repository \(R\): one to send iteration updates, and one to acknowledge the completion of the ATC coordination. The intermediate-level process \(C_{\text{mid}}\) representing design problem \(P_{11}\) receives data of type \(vr \times vy\) from \(C_{\text{top}}\) via channel \(a.0\) and sends data of type \(vr \times vy \times \text{bool}\) via channel \(b.0\). The \(C_{\text{mid}}\) process representing \(P_{11}\) also communicates through channels \(c.0\) and \(d.0\) with the \(C_{\text{bot}}\) process that represents its child \(P_{21}\). Problems \(P_{12}\) and \(P_{22}\) are represented in a similar fashion. The two \(C_{\text{mid}}\) processes and the two \(C_{\text{bot}}\) processes send updates to repository \(R\) through channel arrays \(h\) and \(k\), respectively.

Note that channels \(a\) to \(f\) are channel arrays of size \(ns = 2\). However, \(P_{11}\) has only one child, and therefore only the first elements of the channel arrays \(c\) and \(d\) are used, i.e. \(c.0\) and \(d.0\). The same holds for \(P_{12}\) and channel arrays \(e\) and \(f\). The constant \(ns\) cannot be defined as a separate parameter in the process definition; \(\chi\) is a strongly typed language and does not allow (channel) array type specifications of variable size. At present, \(\chi\) also does not allow to specify the actual array sizes used in a process upon instantiation of the process in a system.

The subproblem optimizations are carried out using the Matlab optimization toolbox function \texttt{fmincon}, which is an implementation of the sequential quadratic programming (SQP) algorithm. We use the square of the \(l_2\)-norm as the metric of the deviation terms in the objective and constraints of Problem (1). The function calls are realized by means of a \(\chi\)-python interface (Hofkamp, 2001), developed such that \(\chi\) can call Python functions like native \(\chi\) functions. Python is a scripting language designed to glue pieces of software to each other. This interface can be used to link \(\chi\) with any other desired software package. In our case, we have used pymat to couple \(\chi\) through Python with Matlab (see Etman et al., 2002).

The system specification is given below:

\[
\begin{align*}
\text{const} & : \text{nat} = 3 \\
mx & : \text{nat} = 1 \\
mb & : \text{nat} = 2 \\
mn & : \text{nat} = 2 \\
nb & : \text{nat} = 2 \\
maxiter & : \text{nat} = 1000 \\
nan & : \text{real} = -9.9e99
\end{align*}
\]
syst. $S(tol, sca : real)$ =
\[
\begin{align*}
[a, c, e : (\neg v \times vy)^{ms}] \\
[b, d, f : (\neg vy \times vy \times bool)^{ms}] \\
[g : \neg vxbtop \times h \times \neg vxbmid]^{nn} \\
[k : \neg vxbtop]^{nn}, s : \text{void}
\end{align*}
\]

\text{Ctop}(P0, a, b, g, s, (0,0), \langle nan, nan, nan \rangle, \langle 1.0 \rangle, \langle 0.0, 0.0 \rangle, \langle sca, sca, 2, 0, 0, 1, 1 \rangle, tol)

\text{Cmid}(P11, a, 0, c, d, h, 0, (1.0), \langle nan \rangle, \langle nan \rangle, \langle 0.0, 0.0 \rangle, \langle sca, sca, 1, 1, 1, 0, 1 \rangle, tol)

\text{Cbot}(P21, e, 0, d, 0, k, 0, \langle 1.0, 1.0, 1.0 \rangle, \langle nan \rangle, \langle nan, 0, 3, 0, 0, 0 \rangle, tol)

\text{R}(g, h, k, s, tol, sca)
\]

(\chi-7)

All processes in system $S$ are instantiated with the appropriate channels and parameters. All optimization variables in the tuple $xh$ of processes $C_{top}$, $C_{mid}$, and $C_{bot}$ are given an initial value of one, except for the $\epsilon_i^n_j$ and $\epsilon_i^n$ variables which are initialized at zero. The parameter array $p$ contains the scaling parameter value $w$ for the $\epsilon$ terms in the objective function of problems $P_0$, $P_{11}$, and $P_{12}$, and the number of children, local design variables, linking variables, and children response and linking variable target values (assumed equal for all children) for each process $C_{top}$, $C_{mid}$, and $C_{bot}$. The $\chi$ specification ends with:

\text{xper} (tol, sca : real) = $[S(tol, sca)]$

(\chi-8)

denoting that after compilation a system execution of $S$ can be carried with tolerance $tol$ and scaling $w$ as input.

4.3 Results

We have implemented three coordination schemes for our three-level target cascading problem. Denoting top, intermediate, and bottom levels with 0, 1, and 2, respectively, the sequences are

Scheme I (downwards only): $0 \rightarrow 1 \rightarrow 2 \rightarrow 0$

Scheme II (downwards/upwards): $0 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow 0$

Scheme III (nested): $0 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow 2 \ldots 0$

It has been observed that scaling the $\epsilon$ terms in the objective of problems 3-5 has an effect on the convergence behavior of the ATC process. Therefore, numerical experiments have been performed with different values for the termination criterion tolerance and scaling parameter $sca$ (same for all $\epsilon$ terms). The results are summarized in Tables 1 - 3. There are two rows for each scaling parameter value; in the top row we report accuracy of obtained results, represented by the square of the $l_2$-norm of the difference between obtained and exact solution. In the bottom row, and for schemes I and II, we report the number of required target cascading iterations. The computational cost can then be determined for each scheme according to the following considerations: For scheme I, the top-level problem is solved $n + 1$ times, where $n$ is the number of target cascading iterations, and each intermediate-level and bottom-level problem is solved $n$ times. Since intermediate-level and bottom-level problems are solved in parallel, the total computational cost in terms of solved optimization problems is therefore $n + 1 + n + n = 3n + 1$. For scheme II, the top level problem is solved $n + 1$ times, each intermediate-level problem is solved $2n$ times, and each bottom-level problem is solved $n$ times; total cost equals $4n + 1$. For scheme III, the bottom row includes three numbers: starting from the left, these are the number of times $n_t$ that the top-level problem has been solved, and the numbers of times $n_l$ and $n_r$ that the left and right intermediate-level and bottom-level problems have been solved, respectively. Note that for each side of the tree (left and right), the number of intermediate-level problems is equal to the number of solved bottom-level problems. The total cost is then given by $n_t + 2 \max(n_l, n_r)$.

The variation of accuracy and computational cost (as defined above) for each coordination scheme is depicted in Figures 5 and 6, respectively, for the termination tolerance value of $10^{-8}$. These plots look quite similar for $tol = 10^{-6}$. However, for $tol = 10^{-10}$, the behavior is quite different, as can be seen in Figures 7 and 8.

Although this example is quite simple and serves, we can draw some interesting conclusions: For low tolerance values, the scaling parameter yielding the highest accuracy differs between scheme I (1e4) and schemes II and III (1e6). For high tolerance values, the scaling parameter yielding the highest accuracy is the same for all schemes (1e4). The accuracy of the obtained results does not improve dramatically as the
Table 1: Results for coordination scheme I

<table>
<thead>
<tr>
<th>sca</th>
<th>Termination tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1e-6</td>
</tr>
<tr>
<td>1e2</td>
<td>0.87526</td>
</tr>
<tr>
<td>1e3</td>
<td>0.03840</td>
</tr>
<tr>
<td>1e4</td>
<td>0.00597</td>
</tr>
<tr>
<td>1e5</td>
<td>0.02541</td>
</tr>
<tr>
<td>1e6</td>
<td>0.02602</td>
</tr>
<tr>
<td>1e7</td>
<td>0.02567</td>
</tr>
<tr>
<td>1e8</td>
<td>0.02595</td>
</tr>
</tbody>
</table>

Table 2: Results for coordination scheme II

<table>
<thead>
<tr>
<th>sca</th>
<th>Termination tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1e-6</td>
</tr>
<tr>
<td>1e2</td>
<td>0.87513</td>
</tr>
<tr>
<td>1e3</td>
<td>0.03825</td>
</tr>
<tr>
<td>1e4</td>
<td>0.03204</td>
</tr>
<tr>
<td>1e5</td>
<td>0.03371</td>
</tr>
<tr>
<td>1e6</td>
<td>0.00863</td>
</tr>
<tr>
<td>1e7</td>
<td>0.01105</td>
</tr>
<tr>
<td>1e8</td>
<td>0.04686</td>
</tr>
</tbody>
</table>

Table 3: Results for coordination scheme III

<table>
<thead>
<tr>
<th>sca</th>
<th>Termination tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1e-6</td>
</tr>
<tr>
<td>1e2</td>
<td>0.87507</td>
</tr>
<tr>
<td>1e3</td>
<td>0.03825</td>
</tr>
<tr>
<td>1e4</td>
<td>0.03205</td>
</tr>
<tr>
<td>1e5</td>
<td>0.01770</td>
</tr>
<tr>
<td>1e6</td>
<td>0.00998</td>
</tr>
<tr>
<td>1e7</td>
<td>0.03014</td>
</tr>
<tr>
<td>1e8</td>
<td>0.04686</td>
</tr>
</tbody>
</table>

Figure 5: Accuracy of results obtained for $tol = 10^{-6}$ as the tolerance value increases for all three schemes, especially considering the substantial increase in computational cost. Overall, although it seems that scheme I provides the most accurate results, scheme III is much more efficient.

5 CONCLUSIONS

The key to the success of the ATC process lies in the coordination. It is advantageous to have a language that enables precise and compact specification of the ATC coordination. The $\chi$ language has been used to specify and implement alternative coordination strategies of the ATC process. Top-level, intermediate-level, and bottom-level processes have been identified for this purpose. Once a specific coordination strategy was implemented, only limited modifications (confined to the intermediate-level process) were required to define additional coordinations. Moreover, larger problems can be treated readily by simply adding necessary instantiations of the defined processes. The example demonstrated that the $\chi$ language is well-suited for representing the coordination schemes.
to specify and implement the ATC coordination, and enables rapid investigation of coordination strategies. The reader should have in mind that the usefulness of $\chi$ has been utilized in a small degree only due to the simplicity of the example. The specification of ATC as parallel processes may lead to new ways of thinking in regard to ATC coordination schemes.

REFERENCES


